P.1877/93

7 1993

informatyka

WYDAWNICTWO SIGMA NOT

Cena 25 000 zł

informatyka

nr 7 1993
Lipiec
Rok wydania XXVIII

Miesięcznik
ISSN 0542-9951
INDEKS 36124

KOLEGIUM REDAKCYJNE:

mgr Jarosław DEMINET
mgr inż. Piotr FUGLEWICZ
mgr Teresa JABŁOŃSKA
(sekretarz redakcji)
Władysław KLEPACZ
(redaktor naczelny)
mgr inż. Jan RYŻKO
dr Zdzisław SZYJEWSKI

PRZEWODNICZĄCY
RADY PROGRAMOWEJ:

Prof. dr hab.
Juliusz Lech KULIKOWSKI

WYDAWCA:
Wydawnictwo Czasopism i Książek Technicznych SIGMA NOT
Spółka z o.o.
ul. Ratuszowa 11
00-950 WARSZAWA
skrytka pocztowa 1004

Redakcja:
01-552 Warszawa,
Pl. Inwalidów 10, p. 104, 105
tel. 39-14-34

Materiałów nie zamówionych redakcja nie zwraca

W sprawach ogłoszeń prosimy zwracać się bezpośrednio do Redakcji lub Działu Reklamy i Marketingu 00-950 Warszawa ul. Mazowiecka 12 telefon: 27-43-66 telefaks: 19-21-87 teleks: 814877

W numerze:

Strona

Emerald – język i system rozproszonego programowania obiektowego – *Przemysław Pardyak, Krzysztof Zieliński* 1
Projektowanie oprogramowania systemów wbudowanych – *Krzysztof M. Sacha* 8
Rynki elektroniczne – podstawowe pojęcia, obszary zastosowań i aspekty ekonomiczne – *Michał Goliński* 16
Systemy wspomagania projektowania w przemyśle reklamowym – *Tomasz Grzelak* 22
Grafika komputerowa w programie studiów – *Jan Zabrodzki* 26

W najbliższych numerach:

- **Tadeusz Witkowski charakteryzuje system modeli komputerowego wspomagania podejmowania decyzji o zakupie produktu w formie przetargu.**
- **Waldemar Wieczerzycki omawia wielowersyjne bazy danych i porównuje je z jednowersyjnymi.**
- **Jarosław Prokop zajmuje się modelowaniem i animacją sylwetek ludzkich oraz opisuje zastosowany system Charlie II, napisany w języku POP-11.**
- **Piotr Porwik charakteryzuje multimedialne bazy danych, przedstawia zasady umieszczania w nich obiektów oraz rejestrowania obrazów analogowych.**
- **Stanisław Wrycza i Tomasz Plata-Przechlewski kontynuują omawianie badań głównych zagadnień projektowania i użytkowania systemów informatycznych.**

Warunki prenumeraty na 1993 r.

Zamówienia na prenumeratę czasopism wydawanych przez Wydawnictwo SIGMA-NOT można składać w dowolnym terminie. Mogą one obejmować dowolny okres czasu, tzn. dotyczyć dowolnej liczby kolejnych zeszytów każdego czasopisma.

Zamawiający może otrzymywać zaprenumerowany przez siebie tytuł począwszy od następnego miesiąca po dokonaniu wpłaty. Zamówienia na zeszyty sprzed daty otrzymania wpłaty będą realizowane w miarę możliwości – z posiadanych zapasów magazynowych.

Warunkiem przyjęcia i realizacji zamówienia jest otrzymanie z banku potwierdzenia dokonania wpłaty przez prenumeratora. Dokument wpłaty jest równoznaczny ze złożeniem zamówienia.

Wpłat na prenumeratę można dokonywać na ogólnie dostępnych blankietach w Urzędach Pocztowych (przekazy pieniężne) lub Bankach (polecenie przelewu), przekazując środki pod adres:
Wydawnictwo SIGMA-NOT Spółka z o.o.
Zakład Kolportażu
00-950 Warszawa, skr. poczt. 1004
konto:
PBK S.A. III 0/Warszawa nr 370015-1573

Wpłaty na prenumeratę od marca br. przyjmują także wszystkie urzędy pocztowe nadawczo-odbiorcze oraz doręczyciele na terenie całego kraju

Na blankiecie wpłaty należy czytelnie podać nazwę zamawianego czasopisma, liczbę zamawianych egzemplarzy, okres prenumeraty oraz własny adres.
Na życzenie prenumeratora, zgłoszone np. telefonicznie, Zakład Kolportażu ul. Bartycka 20, 00-950 Warszawa, (telefony: 40-30-86, 40-35-89 oraz 40-00-21 wew. 249, 293, 299) wysyła specjalne blankiety zamówień wraz z aktualną listą tytułów i cennikiem czasopism.

Odbiorcy zagraniczni mogą otrzymywać czasopisma poprzez prenumeratę dewizową (wpłata dokonywana poza granicami Polski w dewizach, wg cennika dewizowego z cenami podanymi w dolarach amerykańskich) lub poprzez zamówioną w kraju prenumeratę ze zleceniem wysyłki za granicę (zamawiający podaje dokładny adres odbiorcy za granicą, dokonując równocześnie wpłaty w wysokości dwukrotnie wyższej niż cena normalnej prenumeraty krajowej).

Egzemplarze archiwalne (sprzedaż przelewowa lub za zaliczeniem pocztowym) można zamawiać pisemnie, kierując zamówienia pod adresem: Wydawnictwo SIGMA NOT, Spółka z o.o. Zakład Kolportażu, 00-716 Warszawa, ul. Bartycka 20, paw. B, tel. 40-37-31, natomiast za gotówkę można je nabyć w Klubie Prasy Technicznej w Warszawie ul. Mazowieckiej 12, tel. 26-80-17.

Istnieje możliwość zaprenumerowania 1 egz. czasopisma po cenie ulgowej przez indywidualnych członków stowarzyszeń naukowo-technicznych zrzeszonych w FSNT oraz przez uczniów zawodowych i studentów szkół wyższych. Blankiet wpłaty na prenumeratę ulgową musi być opatrzony na wszystkich odcinkach pieczęcią koła SNT lub szkoły.

W przypadku zmiany cen w okresie objętym prenumeratą Wydawnictwo zastrzega sobie prawo do wystąpienia o dopłatę różnicy cen oraz prawo do realizowania prenumeraty tylko w pełni opłaconej.
Cena jednego egzemplarza: normalna 25 000 zł, ulgowa 18 750 zł
Wartość prenumeraty w zł:
Normalna: kwartalna 75 000, półroczna 150 000, roczna 300 000
Ulgowa: kwartalna 56 250, półroczna 112 500, roczna 225 000.

Skład i druk: Drukarnia SIGMA NOT Sp. z o.o. z. 253/93

P. 1877/93

PRZEMYSŁAW PARDYAK
KRZYSZTOF ZIELIŃSKI
Instytut Informatyki AGH
Kraków

Emerald język i system rozproszonego programowania obiektowego

Język Emerald [6] łączy w sobie dwie nowoczesne dziedziny informatyki: programowanie obiektowe i systemy rozproszone. Programowanie obiektowe [8] jest jedną z najszerzej zaakceptowanych technik stosowanych w celu uczynienia programowania bardziej efektywnym i mniej podatnym na błędy. Osiąga się to przez użycie wyższego niż w przypadku tradycyjnych języków strukturalnych poziomu abstrakcji związanego z pojęciem obiektu. Systemy rozproszone [7] udostępniają użytkownikowi zasoby osiągalne za pośrednictwem sieci komputerowych w taki sposób, że komputery połączone siecią są widziane jako jeden spójny system, w którym sieć jest niewidoczna.

Emerald powstał w drugiej połowie lat osiemdziesiątych na University of Washington w Seattle, USA. Jego twórcami są Eric Jul i Norman Hutchinson. Celami projektu Emerald były: umożliwienie efektywnego obiektowego programowania rozproszonego, umożliwienie jednolitego używania operacji lokalnych (w ramach jednego węzła sieci) i zdalnych (między różnymi węzłami sieci) oraz wspomaganie mobilności obiektów. Emerald jest to przede wszystkim oparty na obiektach język ogólnego przeznaczenia. Cecha, która wyróżnia go wśród innych języków tego typu, to bezpośrednie wspomaganie programowania w systemach rozproszonych. Język Emerald jest ściśle związany z rozproszonym systemem Emerald, który jest środowiskiem do wykonywania programów napisanych w tym języku.

Artykuł prezentuje Emerald jako język do obiektowego programowania rozproszonego. Prezentacja ma na celu ukazanie tych jego cech, które wyróżniają go spośród innych tego typu narzędzi. W dyskusji nad Emeraldem pominięto całkowicie problemy implementacyjne, a także strukturę i działanie systemu rozproszonego Emerald. Przyczyną tego jest potrzeba skoncentrowania się na aspektach istotnych dla potencjalnego użytkownika – programisty poszukującego dobrego narzędzia wysokiego poziomu do pisania rozproszonych aplikacji.

Artykuł składa się z dwóch części. Celem pierwszej jest podkreślenie najważniejszych zalet Emeralda przez pokazanie tych problemów, które trudno jest rozwiązać w tradycyjnych językach programowania, a które są łatwo rozwiązywalne w Emeraldzie. Drugą część artykułu stanowi krótka charakterystyka języka, podsumowana przykładem programu.

Dlaczego właśnie Emerald?

W sytuacji, w której istnieje tak wiele różnych języków programowania, tworzenie, uczenie się lub używanie nowego ma sens tylko wtedy, gdy wnosi on coś całkowicie odmiennego, jeśli przed programistą otwiera nowe możliwości. W artykule tym, który stawia sobie za cel prezentację Emeralda wobec szerokiego grona polskich informatyków, chcemy podkreślić te cechy języka, które stanowią nową wartość w informatyce. Wydaje się, iż do cech tych należą: pełna kontrola nad rozproszeniem, jednolitość operacji lokalnych i zdalnych, drobnoziarnista mobilność oraz polimorficzny system typów.

Wspomaganie programowania rozproszonego

Mimo powszechności systemów rozproszonych, programowanie rozproszone jest wciąż postrzegane jako trudne. Istnieją dwie przyczyny takich ocen: po pierwsze, rozproszenie stawia nowe, złożone problemy (np. w dziedzinie synchronizacji czy odporności na błędy), a po drugie, nie istnieje jeszcze odpowiednia metodologia programowania rozproszonego. Podstawową częścią takiej metodologii jest język programowania, który w adekwatny sposób opisuje problemy związane z rozproszeniem.

Emerald powstał w odpowiedzi na potrzebę pisania programów, które bez dodatkowego wysiłku programisty mogłyby się wykonywać w rozproszonym środowisku. Cel ten osiągnięto zapewniając z jednej strony pełną kontrolę programisty nad rozproszeniem programu, a z drugiej strony, pozwalając na pozostawienie rozproszenia niewidocznym, jeżeli nie jest ono istotne w danym zastosowaniu. Dzięki temu podejściu programista może na poziomie języka używać pojęć typu węzeł sieci, lokalizacja obiektu, a więc pojęć bezpośrednio z dziedziny problemu.

Transparentność

Jednym z poważniejszych problemów, które napotyka programista piszący programy mające się wykonywać w środowisku rozproszonym, jest transparentność lokalizacji, czyli ukrycie różnic między operacjami lokalnymi a zdalnymi. Dostęp do zdalnych zasobów jest o wiele bardziej kosztowny niż w przypadku zasobów lokalnych. Dla porównania, lokalne wywołanie procedury napisanej w języku C jest najczęściej formą skoku w ramach tej samej przestrzeni adresowej, podczas gdy zdalne wywołanie procedury (*Remote Procedure Call* – RPC) wymaga najczęściej pośrednictwa jądra systemu operacyjnego oraz przesłania informacji przez sieć. Stawia to problem wyboru między implementacją transparentności a efektywnością. Z jednej strony, należy minimalizować liczbę wywołań zdalnych, a z drugiej strony wymuszanie na programiście, aby przewidział, które wywołania są zdalne jest równoważne z usztywnianiem konfigu-

Pracę badawczą i artykuł wykonano w ramach grantu KBN nr: 8.0077.91.01

racji programu – każda zmiana konfiguracji wymusza poprawienie programu. Tradycyjnie rozważa się dwa rozwiązania tego problemu: zachowuje się transparentność przez wymuszanie zawsze wywołań zdalnych, albo rezygnuje się z transparentności oznaczając każde wywołanie jako zdalne lub lokalne. W przeciwieństwie do tych podejść, Emerald oferuje pełną transparentność przy zachowaniu dużej efektywności, co osiągnięto przez sprawdzanie w czasie wykonania czy wywoływany obiekt jest lokalny czy zdalny.

Najważniejszą cechą wywołań obiektów w Emeraldzie jest to, iż są one niezależne od położenia obiektu (ang. *location independence*). Obiekty są wywoływane zawsze w taki sam sposób, bez względu na to, czy są one lokalne czy zdalne. Co więcej, położenie obiektu w momencie wywołania może być nieznane. Zaletą takiego podejścia jest to, iż identyczny program może być wykonywany w różnych konfiguracjach sieci bez żadnych zmian. Inną możliwością jest korzystanie z transparentnego rozpraszania: obiekty składające się na program mogą być automatycznie rozesłane w sieci (np. w celu zrównoważenia obciążenia węzłów sieci) bez wiedzy czy ingerencji programisty.

Mobilność

Rozproszenie dodaje do programowania nowy wymiar: lokalizację obliczenia (modułu, obiektu). Położenie obiektu może być statyczne lub dynamiczne. W pierwszym przypadku, obiekt pozostaje w miejscu utworzenia przez cały okres swego istnienia. W drugim przypadku, obiekt może się przemieszczać między węzłami sieci. Zasadniczą zaletą systemów pozwalających na przemieszczanie obiektów jest możliwość dynamicznej zmiany konfiguracji programu. Możliwość ta jest szczególnie istotna w systemach, od których wymaga się podwyższonej odporności (ang. *robustness*). Dla przykładu, odpowiednia reakcja systemu na wyłączenie węzła sieci jest prostsza, gdy wszystkie działające tam obiekty można przenieść do innego węzła bez przerywania ich pracy.

Większość systemów rozproszonych nie pozwala na jakąkolwiek zmianę położenia obiektów. Z tych, które dopuszczają takie zmiany, większość nakłada na przemieszczane obiekty różnorakie ograniczenia. Typowo – na przykład – można przemieszczać tylko obiekty mające własną przestrzeń adresową (np. proces UNIX-a), co wiąże się z dużym rozmiarem obiektów oraz wysokim kosztem dokonywanych na nich operacji. Innym typowym ograniczeniem jest wymaganie, aby obiekt był nieaktywny (bez działającego w nim procesu). Emerald jest pierwszym systemem, który nie ma tych ograniczeń i dopuszcza w pełni drobnoziarnistą mobilność obiektów (ang. *fine-grained mobility*). Każdy obiekt, bez względu na jego rozmiar, może być swobodnie przemieszczany w sieci. Emerald pozwala również na przesyłanie obiektów razem z działającymi w nich procesami.

Polimorficzny system typów

Jednym z aspektów, które określają wartość języka programowania jest efektywność tworzenia w nim programów. Wykorzystanie kodu (ang. *code reuse*) pozwala na tworzenie oprogramowania z gotowych komponentów, co przyspiesza proces pisania i zwiększa niezawodność oprogramowania. Jedną z umożliwiających to metod jest polimorfizm, czyli zdolność elementów występujących w języku do przyjmowania wielu różnych form. Typowym przykładem jest tworzenie takich obiektów, jak ogólny (ang. *generic*) stos lub lista. Taki ogólny stos nie ma określonego typu elementów, które mogą być w nim przechowywane, i nie może być bezpośrednio użyty w charakterze stosu. Jest on jednak wzorcem konkretnych stosów i może być wykorzystany do stworzenia stosu elementów dowolnego typu, na przykład wartości całkowitych. Interesujące jest przy tym, że stworzenie wyspecjalizowanego stosu nie wymaga żadnego programowania – operacje na stosie są wyspecyfikowane raz i nie zmieniają się, bez względu na typ użytych w konkretnym stosie elementów.

Obiekt może być zawsze zastąpiony innym obiektem o tym samym typie. W językach, które (jak Pascal) przypisują wyrażeniom dokładnie jeden typ oznacza to, iż tylko obiekty o identycznym typie mogą być używane zamiennie. Wprowadzenie polimorfizmu, opartego na zgodności typów oraz separacja między typem a reprezentacją obiektu, umożliwia swobodne zastępowanie obiektów innymi, o ile tylko nowy obiekt jest w stanie spełnić wszystkie funkcje wyjściowego obiektu. Dzięki temu można tworzyć modularne oprogramowanie, którego części mogą być zastępowane innymi, oferującymi na przykład niższy koszt operacji lub podwyższony poziom niezawodności (ang. *reliability*).

System typów Emeralda ma obie wyżej wymienione zalety. Jest on polimorficzny, czyli umożliwia tworzenie ogólnych obiektów oraz ich późniejsze specjalizowanie. Jest on oparty na typach abstrakcyjnych i relacji zgodności między nimi, co pozwala na swobodne zastępowanie obiektów innymi, funkcjonalnie równoważnymi.

Cechy języka Emerald

W rozdziale tym zostaną omówione najistotniejsze cechy języka Emerald, przedstawiając najpierw model obiektu w Emeraldzie i pokazując programowanie w tym języku. Następnie zostanie omówiony pokrótce system typów Emeralda. Na zakończenie zostanie pokazane, w jaki sposób Emerald umożliwia kontrolowanie rozproszenia, jak można tworzyć trwałe obiekty oraz mechanizm komunikacji grupowej w Emeraldzie.

Obiekty w Emeraldzie

Cecha, która odróżnia programowanie obiektowe od programowania strukturalnego, to ścisłe związanie procedur z danymi, na których one operują. Obiekt definiuje się jako pewną strukturę danych wraz z procedurami umożliwiającymi manipulacje na tej strukturze. W typowym języku zorientowanym obiektowo (np. Smalltalk [3]), dane należące do obiektu są dostępne na zewnątrz obiektu tylko za pośrednictwem procedur należących do tego obiektu. Jedyną metodą dostępu do obiektu jest wywołanie (ang. *invocation*) procedury w nim zdefiniowanej. To zamknięcie (ang. *encapsulation*) obiektu umożliwia oddzielenie implementacji obiektu (tj. zastosowanych struktur danych i wnętrza procedur) od jego interfejsu (zbioru operacji dostępnych w obiekcie). Dzięki temu programista jest uwolniony od tego, „jak" obiekt jest zdefiniowany i może skoncentrować się na tym, „co" dany obiekt potrafi.

Emerald jest językiem całkowicie opartym na obiektach. Każdy obiekt Emeralda zawiera:
- prywatne dane (ang. *private data*),
- prywatne operacje (ang. *private operations*),
- publiczne operacje (ang. *public operations*).

Dane zawarte w obiekcie oraz prywatne operacje są na zewnątrz, niedostępne. Jedynie publiczne operacje są widoczne i określają interfejs obiektu.

Struktura obiektu. Obiekt w Emeraldzie [1] ma cztery składowe:
nazwę – unikalny identyfikator obiektu w sieci,
reprezentację – opis danych zawartych w obiekcie,
zbiór operacji – które mogą być wywołane na obiekcie oraz

proces – który wykonuje się w obiekcie niezależnie od wywołanych w nim operacji.

Dodatkowo każdy obiekt w Emeraldzie może mieć dwie opcjonalne sekcje, które umożliwiają wyspecyfikowanie akcji podejmowanych przez obiekt zanim rozpocznie on swe właściwe działanie, tj. zanim powstanie możliwość jego wywoływania przez inne obiekty i zostanie uruchomiony jego proces. Sekcja inicjująca (*initially*) jest wykonywana podczas tworzenia obiektu. Sekcja odtwarzania (*recovery*) służy do odtwarzania obiektu po awarii węzła, w którym obiekt się znajduje.

W związku z tym, iż programy napisane w Emeraldzie wykonują się w sieci, obiekty składające się na program mogą znajdować się w różnych węzłach. Położenie (ang. *location*) obiektu jest więc jego istotnym atrybutem. Początkowo obiekt znajduje się w węźle, w którym został stworzony. W czasie swojego istnienia może on być przemieszczany w sieci, zmieniając swe położenie.

```
const Example ==
  object Example
    export Oper1, Fun1
    const C1 : String
    var V1 : Integer
    monitor
      const C2 : Integer
      var V2 : Boolean
      operation Oper1[arg : Integer]
        ...
      end Oper1
      initially
        ...
      end initially
      recovery
        ...
      end recovery
    end monitor
    operation Oper2 → [res1 : Boolean, res2 : Integer]
      ...
    end Oper2
    function Fun1[arg : Integer] → [res : Integer]
      ...
    end Fun1
    process
      ...
    end process
  end Example
```

Rys. 1. Przykładowy obiekt

Innym istotnym atrybutem obiektów w Emeraldzie jest ich zmienność. Mówimy, że obiekt jest zmienny (ang. *mutable*), jeżeli jego stan (prywatne dane, które w sobie przechowuje) może się zmienić. Obiekt jest niezmienny (ang. *immutable*), jeżeli jego stan nie może być zmieniony. Zaletą używania niezmiennych obiektów jest łatwość ich używania w środowisku rozproszonym. Jeżeli obiekt nie może się zmienić, to można go powielić we wszystkich węzłach sieci i wszędzie używać jak obiekt lokalny.

Tworzenie nowych obiektów. W językach obiektowo zorientowanych, które mają klasy, obiekty są tworzone przez odwołanie się do klasy w celu stworzenia nowego jej przedstawiciela (ang. *instance*). Alternatywą dla mechanizmu klas są tak zwane prototypy (ang. *prototype*) – obiekty, z których przez klonowanie i modyfikacje uzyskuje się nowe obiekty. Emerald nie ma ani klas, ani prototypów. Jedynym dostępnym sposobem tworzenia obiektów są wyrażenia obiektowe (ang. *object constructors*).

Wyrażenie obiektowe spełnia analogiczną rolę jak klasy w językach opartych na klasach. Po pierwsze, umożliwia ono tworzenie nowych obiektów. Po drugie, zawiera pełny opis danych w nim zawartych w obiekcie. Po trzecie, daje opis kodu implementującego operacje dostępne na obiektach przezeń utworzonych. Wyrażenia obiektowe, to kod źródłowy objęty nawiasami syntaktycznymi **object** *Identifier* i **end** *Identifier*. Zawiera ono: listę publicznych operacji (**export**), definicje operacji i funkcji, deklaracje zmiennych i stałych oraz definicje procesu i sekcji pomocniczych. Obiekty są tworzone przez ewaluację wyrażenia obiektowego. Każda kolejna ewaluacja tego samego wyrażenia powoduje stworzenie nowego obiektu. Kolejne obiekty, tworzone przez to samo wyrażenie obiektowe, nie muszą być identyczne – zmienne lokalne, zewnętrzne względem wyrażenia, stają się stałymi w nowo stworzonym obiekcie.

Jednolity model obliczeniowy. Emerald dostarcza jednolitego modelu obliczeniowego do operacji lokalnych i zdalnych. Jedynymi używanymi pojęciami jest obiekt oraz wywołanie obiektu, przy czym oba te pojęcia są niezależne od rozproszenia. Ten sam model obiektu jest używany w Emeraldzie zarówno w przypadku obiektów lokalnych, jak i dla obiektów dostępnych przez sieć, co powoduje, iż obiekty mogą być tworzone w sposób jednolity, niezależnie czy będą one wywoływane lokalnie czy zdalnie. Również sposób wywoływania obiektów jest niezależny od ich położenia: nie trzeba zaznaczać wywołań zdalnych, każde wywołanie może okazać się w czasie wykonania lokalnym lub zdalnym.

Programowanie współbieżne

Emerald umożliwia tradycyjne programowanie współbieżne. Prawdziwą równoległość zapewnia wykonywanie procesów w różnych węzłach sieci. W ramach każdego węzła jest zapewnione współbieżne wykonywanie procesów związanych z obiektami, które się w nim znajdują. Procesy mogą się wykonywać współbieżnie również w ramach obiektu: własny proces oraz procesy, które wykonują zdefiniowane w obiekcie operacje, mogą przebywać w obiekcie jednocześnie.

Obiekty w Emeraldzie mogą być aktywne, tzn. mogą one zawierać własny wątek wykonania (ang. *thread of execution*) oraz dzięki temu podejmować własne akcje, niezależne od akcji innych obiektów. Jest to możliwe dzięki procesorowi (**process**), który może być zdefiniowany w obiekcie. W momencie tworzenia (lub odtwarzania) obiektu, tworzony jest nowy wątek, który wykonuje najpierw sekcję inicjującą (lub odtwarzającą), a następnie proces.

Obiekty komunikują się między sobą za pomocą wywołań. Wywołanie operacji w obiekcie jest synchroniczne – proces wywołujący operację jest zablokowany aż do chwili ukończenia wykonywania operacji i zwrócenia wyników.

Do zapewnienia kontroli nad współbieżnością w ramach obiektu służą monitory. Każdy obiekt składa się z dwóch części: objętej monitorem i nie objętej monitorem. Każda z tych części może być pusta lub może zawierać dane i operacje. Jednocześnie tylko jedna operacja umieszczona w monitorze może być wykonywana. Dowolna liczba procesów może wykonywać jednocześnie operacje nie umieszczone w monitorze. Zmienne globalne w ramach obiektu mogą być modyfikowane tylko

przez operacje monitorowane i to wyłącznie wtedy, jeśli same są zdefiniowane w części objętej monitorem. Wyjątkiem są zmienne globalne zdefiniowane poza monitorem, które mogą być inicjowane przez sekcję inicjującą lub odtwarzającą. Synchronizacja w ramach monitora jest osiągana za pomocą obiektów-warunków (ang. *condition objects*). Proces znajdujący się w monitorze może wykonać operację **wait** na takim obiekcie, zawieszając swe wykonanie i opuszczając monitor. Zawieszony proces może zostać odblokowany przez wykonanie operacji **signal** na odpowiednim obiekcie-warunku.

```
const myAbstractType ==
  type myAbstractType
    operation Funny → [Integer]
    function Cosy[String] → [Integer]
    operation Hairy[Integer,AbstractType] → [AbstractType,String]
  end myAbstractType
```

Rys. 2. Przykładowy typ abstrakcyjny

System typów

Emerald ma silny system typów (ang. *strongly typed language*) [2], to znaczy, że każde wyrażenie ma typ, który może być określony na etapie kompilacji. Silny system typów zapewnia całkowicie statyczną kontrolę typów. Typy wszystkich wyrażeń są sprawdzane podczas kompilacji, dzięki czemu w czasie wykonania nie występuje potrzeba sprawdzania typów, a podczas wykonania nigdy nie występują błędy w typie.

Typy abstrakcyjne. Typy w Emeraldzie są abstrakcyjne, to znaczy, że typ określa wyłącznie operacje dostępne na obiekcie i nie mówi nic na temat ich implementacji. Typ jest zbiorem sygnatur operacji składających się z nazwy (identyfikatora) operacji oraz liczby i typów jej argumentów i wyników. Na przykład, typ *myAbstractType* przedstawiony na rys. 2 zawiera sygnatury trzech operacji. W nawiasach przed strzałką podane są typy argumentów operacji, a po strzałce typy wyników. Mówimy, że obiekt jest typu *T*, jeżeli implementuje on wszystkie operacje zdefiniowane w tym typie.

Jak już wspomniano, obiekty są w Emeraldzie jedynymi istniejącymi jednostkami. Również typy są obiektami oraz istnieje wyróżniony typ *AbstractType*, który jest typem typów. Obiekt typu *AbstractType* (a więc typ), ma jedną operację o nazwie *GetSignature*, która jako wynik zwraca sygnatury wszystkich operacji zdefiniowanych w danym typie.

Zgodność typów (ang. *type conformance*) jest kluczowym pojęciem w języku Emerald. Dwa typy mogą być zgodne, co powoduje, iż obiekty należące do jednego typu mogą zastąpić obiekty drugiego typu. Dzięki temu jest możliwe zastępowanie obiektów innymi, o nieznanej implementacji. Autorzy języka w następujący sposób definiują zgodność typów:

„Nieformalnie, typ *S* jest zgodny (ang. *conforms*) z typem *T* (co zapisuje się $S \circ\!\!> T$) jeżeli obiekt typu *S* może zawsze zastąpić obiekt typu *T*, to znaczy, obiekt typu *S* może być zawsze użyty w miejscu, gdzie oczekiwany jest obiekt typu *T*. Aby obiekty typu *S* mogły zastępować obiekty typu *T* wymagane jest, aby:
1. *S* zawierał co najmniej operacje występujące w *T* (*S* może mieć więcej operacji),
2. Dla każdej operacji w *T*, odpowiadająca jej operacja w *S* miała tą samą liczbę argumentów i wyników,
3. Typy wyników operacji występujących w *S* były zgodne z typami wyników operacji występujących w *T*,
4. Typy argumentów operacji występujących w *T* zgodne z typami argumentów operacji występujących w *S*".

Relacja zgodności typów tworzy porządek częściowy na zbiorze typów języka Emerald. Element (typ) większy w tym porządku może zawsze zastąpić element mniejszy. Najmniejszym elementem w tym uporządkowaniu jest typ, który nie ma żadnej operacji. Ten typ nazywa się *Any* i każdy typ jest z nim zgodny. Wszystkie obiekty są typu *Any*, ponieważ ten typ nie stawia żadnych wymagań co do operacji implementowanych przez obiekt (najprościej rzecz ujmując: każdy obiekt ma co najmniej zero operacji). Największym elementem w tym uporządkowaniu jest typ, który zawiera wszystkie możliwe operacje. Ten typ nazywa się *None* i jest zgodny z wszystkimi możliwymi typami. Istnieje tylko jeden wyróżniony obiekt (*NIL*) należący do typu *None*. Ten obiekt może być przypisany do zmiennej dowolnego typu i oznacza niezainicjowaną zmienną.

Polimorfizm. W Emeraldzie polimorfizm występuje na kilku poziomach. Po pierwsze, dzięki relacji zgodności obiekty są polimorficzne. Jest tak dlatego, że obiekt ma tyle typów, z iloma jest on zgodny. Po drugie, operacje definiowane na obiektach są polimorficzne, ponieważ obiekty różnych typów mogą być dzięki zgodności przekazane jako parametry wywołania danej operacji. Po trzecie, dzięki temu, że typy są obiektami, operacje mogą przyjmować typy jako parametry i zwracać typy jako wyniki. Umożliwia to tworzenie polimorficznych obiektów takich, do których można przekazać typ jako parametr, aby utworzyć nowy wyspecjalizowany obiekt. Na przykład, tablice w Emeraldzie są implementowane za pomocą obiektu *Array*. Obiekt ten ma jedną operację *Of*, która przyjmuje jako parametr typ podstawowy przyszłej tablicy. Obiekt utworzony za pomocą wywołania *Array.Of*[*Integer*] służy do tworzenia za pomocą operacji *Create* tablic o typie podstawowym *Integer*. Tak więc obiekt *Array.Of*[*Integer*].*Create* [13] jest trzynastoelementową tablicą o typie całkowitym.

Kontrola rozproszenia

Najbardziej wyróżniającą cechą Emeralda jest zespół konstrukcji do wspomagania programowania rozproszonego [4]. Konstrukcje te umożliwiają sprawdzenie położenia obiektu, jego przemieszczenie oraz sprawdzenie stanu systemu rozproszonego.

Opis systemu rozproszonego. Emerald umożliwia obiektom sprawdzenie stanu systemu rozproszonego, w którym one działają. Podstawowym pojęciem jest tutaj węzeł sieci. Węzły są reprezentowane przez specjalne obiekty typu *Node*. Każdemu fizycznemu węzłowi odpowiada jeden obiekt typu *Node*. Obiekt taki umożliwia dokonanie operacji związanych ściśle z reprezentowanym komputerem. Przykładowo, można sprawdzić lokalny czas lub obciążenie systemu. Dodatkowo można zażądać informacji na temat stanu całego systemu. Odpowiednie operacje zwracają w wyniku ich wywołania listę fizycznych węzłów sieci lub tych z nich, które są aktywne (działa w nich jądro Emeralda). Bardzo istotną możliwością dostarczaną przez obiekt typu *Node* jest monitorowanie zmian stanu systemu. Można zażądać, aby dany obiekt był informowany (odpowiednia, w nim wywołana operacja) ilekroć nowy węzeł stanie się aktywny, zostanie wyłączony lub przez dłuższy czas nie odpowiada na próby komunikacji.

Lokalizacja obiektu. Każdy obiekt może być zlokalizowany w sieci za pomocą wyrażenia **locate**. Wyrażenie to zwraca jako wynik obiekt typu *Node* odpowiadający bieżącemu położeniu badanego obiektu.

```
var aNode     : Node
var anObject  : Any
aNode <- locate anObject
```

Mobilność. Istnieje kilka poleceń służących do przesyłania obiektów między węzłami sieci. Podstawowym poleceniem jest zdanie **move**.

move anObject to aNode

Zdanie **move** nie jest żądaniem, ale wskazówką dla systemu. Może być ono zignorowane lub jego realizacja odłożona na później. Zdanie to może służyć do przekazywania sugestii o pożądanym położeniu obiektu, np. do modułu równoważącego obciążenie systemu rozproszonego.

Aby wymusić zmianę położenia obiektu należy użyć zdania **fix**.

fix anObject to aNode

Zdanie **fix** powoduje, iż po przemieszczeniu położenie obiektu staje się ustalone i nie może być zmienione za pomocą zdań **move** i **fix**.

Aby zmienić położenie obiektu, o ustalonym za pomocą **fix** położeniu, należy użyć zdania **unfix**, które nie zmieniając położenia obiektu odblokowuje go umożliwiając dalsze przemieszczanie. Dodatkowe zdanie **refix** jest nierozdzielnym połączeniem zdań **unfix** i **fix**. Umożliwia ono zmianę położenia obiektu bez uwalniania go ze stanu „ustalenia".

Zdarza się, że grupa obiektów jest ze sobą ściśle związana, np. współdziałają one przy wykonywaniu swoich zadań. W przypadku mobilności tak związanych obiektów, przemieszczenie tylko części z nich powoduje zwiększenie kosztów wykonywanych na nich operacji. Aby umożliwić automatyczne przesyłanie całej grupy obiektów, w Emeraldzie wprowadzono pojęcie związanych obiektów (ang. *attached*). Każda zmienna lub stała wewnątrz obiektu może być opatrzona słowem kluczowym **attached**. W przypadku przesyłania obiektu zawierającego taką deklarację, automatycznie zostanie przesłany obiekt przypisany do takiej zmiennej lub stałej.

Przekazywanie parametrów. Unikalną cechą języka Emerald są dwa nowe tryby przekazywania parametrów, specjalnie dostosowane do programowania w środowisku rozproszonym. Umożliwiają one przesyłanie parametrów wywołania obiektu do węzła, w którym znajduje się wywoływany obiekt.

Jeżeli parametr jest poprzedzony słowem kluczowym **move**, to mamy do czynienia z przekazywaniem parametru „z przemieszczeniem" (ang. *call-by-move*). Parametr taki zostanie przesłany do odpowiedniego węzła, zanim operacja zostanie wywołana. Drugi tryb, to przekazywanie parametru „z wizytą" (ang. *call-by-visit*). Parametr poprzedzony słowem kluczowym **visit** zostaje przesłany do odpowiedniego węzła przed wywołaniem operacji, a po jej zakończeniu zostanie odesłany z powrotem do wyjściowego węzła.

Celem zastosowania tych dwóch nowych trybów jest optymalizacja kosztu zdalnego wywołania obiektu. W Emeraldzie przekazywanie parametrów wywołania operacji polega na przekazaniu identyfikatorów odpowiednich obiektów (nie następuje kopiowanie obiektów). W przypadku wywołania zdalnego oznacza to, iż każde odwołanie do parametru wewnątrz wywołanej operacji jest również zdalne, o ile wywołany obiekt i parametr nie znajdują się w tym samym węźle. Przesłanie parametru razem z wywołaniem podnosi efektywność programu, ponieważ zmniejsza się liczba zdalnych wywołań przez umieszczenie parametru w węźle, z którego będzie wywoływany.

Trwałe obiekty

Często jest istotne, aby fragmenty systemu rozproszonego były w stanie przetrwać awarię węzła, w którym się znajdują. Przykładowo obiekt, który zarządza lokalnym zasobem, powinien mieć możliwość, odtworzenia swego stanu sprzed awarii, aby móc podjąć niezakłóconą pracę w momencie powrotu węzła do działania. Istnieje wiele technik rozwiązywania tego problemu; jedna z prostszych polega na zapisie stanu całego obiektu na żądanie użytkownika. Z tym właśnie schematem mamy do czynienia w Emeraldzie. Każdy obiekt przez wywołanie zdania **checkpoint**, może zostać zapisany na dysk. Przy następnym uruchomieniu jądra Emeralda w węźle, w którym to nastąpiło, obiekt ten zostanie odtworzony z dysku oraz zostanie w nim uruchomiona sekcja odtwarzania. Przez odpowiednie zaprogramowanie obiektu oraz zapisywanie go na dysk po każdej zmianie jego stanu, można osiągnąć w Emeraldzie płynną, mimo awarii węzłów, pracę obiektów.

Komunikacja grupowa

Ze względu na liczbę uczestników, każdy akt komunikacji można zaliczyć do jednej z dwóch kategorii: prostej komunikacji między dwoma uczestnikami oraz komunikacji grupowej, która może obejmować dowolną liczbę uczestników. W informatyce prosta komunikacja to np. wywołanie procedury w jakimś module lub przekazanie sygnału z jednego procesu do drugiego. Przykładem komunikacji grupowej jest *broadcast* – przesłanie komunikatu od jednego komputera do wszystkich innych komputerów w sieci. Często zdarza się, że w systemach składających się z wielu współpracujących modułów, większa liczba modułów realizuje to samo zadanie lub współzawodniczy

```
const NameChannel←Channel.Of[ServerType].Create   % stworzenie nowej grupy
NameChannel.Subscribe[Server1]                    % dodanie obiektu do grupy
NameChannel.UnSubscribe[Server9]                  % usuniecie obiektu z grupy
broadcast LookUp[Name] on NameChannel             % wywolanie grupy
   to Receivers : Array.Of[ServerType]            % lista czlonkow grupy w momencie wywolania
   R ← NIL
   quit when Receivers.Empty
   reply Item : Any received                      % przetwarzanie wynikow
      last isLast : Boolean                       % flaga ustawiana gdy otrzymano ostatni wynik
      if Item !== NIL then
         R ← Item
         quit
      end if
      quit when isLast
   end received
end broadcast
```

Rys. 3. Przykład operowania grupami

o jakiś zasób. W takich sytuacjach okazuje się, że system zyskuje na prostocie i efektywności, jeżeli zastosuje się komunikację grupową.

Komunikacja grupowa może być efektywnie zastosowana w systemach obiektowych, gdyż mogą być one widziane jako zespoły niezależnych obiektów komunikujących się za pomocą wywołań. W wyniku badań prowadzonych na Uniwersytecie w Kopenhadze, Emerald został rozszerzony o mechanizmy komunikacji grupowej między obiektami [5]. Te mechanizmy to, po pierwsze tworzenie i modyfikacje grup obiektów, a po drugie, możliwość wywołania takiej grupy. Na rys. 3 pokazano typowe wykorzystanie grupy obiektów. Obiekt *Channel* służy do tworzenia grup. W wyniku wywołania operacji *Of*[*ServerType*].*Create* zostanie utworzona grupa, do której mogą dołączać obiekty typu *ServerType*. Początkowo grupa jest pusta, nie zawiera żadnych obiektów. Obiekty mogą być dołączane do grupy za pomocą operacji *Subscribe* i usuwane za pomocą operacji *UnSubscribe*. Grupa jest wywoływana za pomocą zdania **broadcast**. W wyniku wykonania takiego zdania z rys. 3, operacja *LookUp* zostanie wywołana we wszystkich obiektach, które są członkami grupy identyfikowanej przez *NameChannel*. Każdy z tych obiektów zwraca wynik wywołania tej operacji, który może być przechwycony i wykorzystany w sekcji **reply**. Sekcja **reply** jest wykonywana dla każdego wyniku osobno. Zbieranie wyników może być przerwane w dowolnym momencie przez wykonanie zdania **quit**.

```
const Element ==
   immutable object Element
      export GetSignature, Create
      function GetSignature → [Result : Signature]
         Result ←
            type ElementType
               operation OneMore[Integer, ElementType]
            end ElementType
      end GetSignature
      operation Create[nb : Integer, next : ElementType] → [Result : ElementType]
         Result ←
            object anElement
               export OneMore
               const myNode ← locate self
               const myOut ← myNode.GetStdOut
               const Elements ← Array.Of[ElementType].Empty
               monitor
                  operation OneMore[i : Integer, new : ElementType]
                     myOut.PutString[nb.AsString || ": new " || i.AsString || "\n"]
                     if next !== NIL then
                        next.OneMore[i, new]
                     end if
                     Elements.AddUpper[new]
                  end OneMore
                  initially
                     if next !== NIL then
                        Elements.AddUpper[next]
                        next.OneMore[nb, self]
                     end if
                  end initially
               end monitor
            end anElement
      end Create
   end Element
const Test ==
   object Test
      const myNode ← locate self
      const ActiveNodes ← myNode.GetActiveNodes
      process
         var i, lb, ub : Integer
         var newElement : Element ← NIL
         lb ← ActiveNodes.LowerBound
         ub ← ActiveNodes.UpperBound
         StdOut.PutString[(ub−lb+1).AsString || " active nodes detected\n"]
         for( i←lb : i <= ub : i←i+1 )
            move self to ActiveNodes(i).GetTheNode
            newElement ← Element.Create[i,newElement]
         end for
         move self to myNode
         StdOut.PutString["System booted\n"]
      end process
   end Test
```

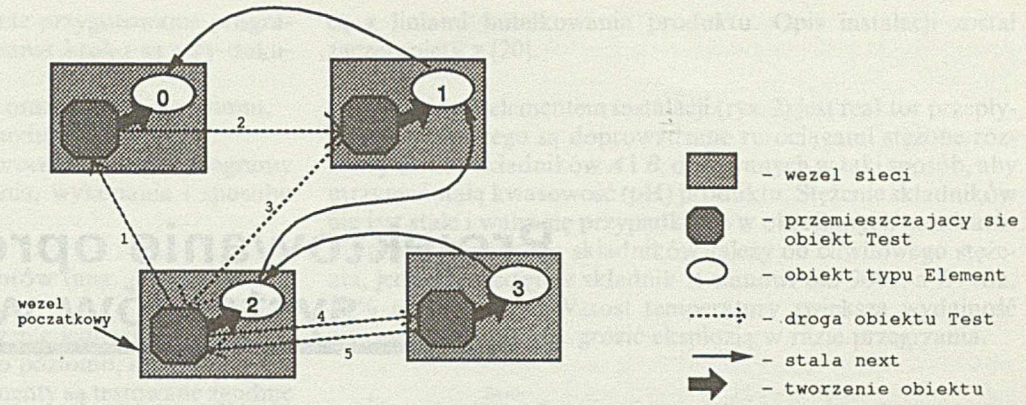
Rys. 4. Przykład programu

Przykładowy program

Na rys. 4 przedstawiono przykładowy prosty program w Emeraldzie. Program w Emeraldzie składa się z pewnej liczby definicji stałych. Wartości przypisane tym stałym są obliczane w momencie uruchomienia programu. W szczególności, jeżeli wyrażeniem tym jest wyrażenie obiektowe, nowy obiekt zostaje utworzony, a jego proces uruchomiony. Nasz przykładowy program składa się z dwóch definicji stałych. Jego uruchomienie spowoduje utworzenie dwóch obiektów: *Element* i *Test*. Pierwszy obiekt spełnia podwójną rolę. Służy do tworzenia za pomocą operacji *Create* obiektów, dla których jest jednocześnie typem abstrakcyjnym. Drugi obiekt zawiera w sobie proces, który tworzy pewną liczbę obiektów typu *Element*.

W wyniku wykonania naszego przykładowego programu zostanie utworzony system porozumiewających się identycznych obiektów. System ten może być traktowany jako szkielet do budowy rozproszonych serwisów, np. serwisu nazw (ang. *distributed name service*). Obiekt *Element* jest wykorzystywany do tworzenia pojedynczych elementów tego systemu, podczas gdy obiekt *Test* służy do uruchomienia systemu. System składa się z obiektów typu *Element*. W każdym węźle sieci zostaje uruchomiony jeden taki obiekt. Wszystkie elementy systemu są połączone w łańcuch, więc każdy z nich może komunikować się z pozostałymi.

Wykonanie programu składa się z dwóch faz. W pierwszej z nich obiekt *Test* uruchamia system, w drugiej system działa już niezależnie. Obiekt *Test*



Rys. 5. Konfiguracja systemu utworzonego przez przykładowy program

najpierw sprawdza listę działających w sieci węzłów (operacja *GetActiveNodes*). Następnie w pętli w każdym z tych węzłów tworzy obiekt typu *Element*. W tym celu przenosi się do kolejnego węzła, gdzie tworzy nowy obiekt. Każdy nowo utworzony obiekt otrzymuje informację o swoim numerze seryjnym oraz o następnym łańcuchu w obiekcie. Wzdłuż tego łańcucha za pomocą operacji *OneMore* jest przekazywana do wszystkich części systemu informacja o następnym elemencie. Druga faza pracy systemu nie jest uwidoczniona w programie. Gdyby obiekty typu *Element* miały proces lub operacje inne niż *OneMore*, to mogłyby wykonywać jakieś pożyteczne zadania. Rys. 5 przedstawia efekt wykonania pierwszej fazy programu. Uwidoczniono aktywne węzły, trasę obiektu *Test* oraz utworzone obiekty typu *Element*.

W naszym programie pokazano kilka typowych schematów pisania programu w Emeraldzie. Obiekt *Element* jest typowym kreatorem: obiektem służącym do tworzenia innych obiektów. Zdefiniowana w tym obiekcie operacja *Create* jest typowym przykładem wykorzystania wyrażeń obiektowych. Operacja ta jest wywołana przez obiekt *Test* i zwraca jako swój wynik nowy obiekt typu *Element*. Każdy utworzony przez operację obiekt ma dwie stałe: *nb* oraz *next* o wartości równej wartości argumentów operacji *Create* w momencie tworzenia obiektu. Operacja *GetSignature* sprawia, iż *Element* jest jednocześnie typem abstrakcyjnym dla tworzonych przez siebie obiektów. Sposób, w jaki jest tworzony system obiektów typu *Element* jest również typowy. Obiekt rozpraszający ten system najpierw sprawdza stan systemu, a następnie obiega aktywne węzły i tworzy w nich odpowiednie obiekty.

* * *

W istniejącej implementacji Emerald może być z powodzeniem używany w dwóch celach: edukacyjnym i eksperymentalnym. Nowoczesność języka w zakresie systemów rozproszonych oraz łatwość panowania nad rozproszeniem powodują, iż może być on zastosowany w ramach kursu programowania rozproszonego[1]. Innowacyjny system typów oraz nietradycyjny dla języków obiektowych sposób programowania, pozwalają używać Emerald jako interesujący drugi (np. po Smalltalku) język na kursie programowania obiektowego. Emerald może być również wykorzystany jako podstawa do eksperymentów w dziedzinie obiektowo zorientowanych systemów rozproszonych. Jego względnie mały rozmiar oraz fakt, że system Emerald jest uruchomiony jako proces UNIX-a powodują, że system i język mogą być łatwo modyfikowane lub używane do testowania własnych pomysłów w tej dziedzinie. Jako przykład, można podać badania prowadzone przez jednego z autorów w dziedzinie komunikacji grupowej, w których Emerald posłużył do zaimplementowania nowych konstrukcji.

Emerald jest dostępny przez FTP w kilku węzłach sieci Internet – może być swobodnie kopiowany i używany do celów edukacyjnych lub naukowych. Głównym punktem dystrybucji Emeralda w Europie jest Wydział Informatyki Uniwersytetu Kopenhaskiego (węzeł *diku.dk*). Pytania dotyczące systemu i języka mogą być kierowane przez pocztę elektroniczną pod adresem
emerald@diku.dk. lub pardy@tulip.ics.agh.edu.pl.

[1] Kurs taki, z wykorzystaniem systemów PVM, SR i Emerald, jest prowadzony na piątym roku studiów w Instytucie Informatyki AGH

LITERATURA

[1] Black A., Hutchinson N., Jul E., Levy H.: Object Structure in the Emerald System. In: Proceedings of the 1986 ACM Conference on Object-Oriented Programming Systems, Languages and Applications, pp. 78–86, October 1986

[2] Black A., Hutchinson N., Jul E., Levy H., Carter L.: Distribution and Abstract Types in Emerald. IEEE Transactions on Software Engineering, 13 (1), pp. 65–76, January 1987

[3] Goldberg A., Robson D.: Smalltalk-80: The Language And Its Implementation. Addison-Wesley Publishing Company, 1983

[4] Jul E., Levy H., Hutchinson N., Black A.: Fine-Grained Mobility in the Emerald System. ACM Transactions on Computer Systems, 6 (1), pp. 109–133, February 1988

[5] Pardyak P.: Group Communication in an Object-Based Environment. In: Proceedings of the 2nd International Workshop on Object-Orientation in Operating Systems I-WOOOS'92, Paris, France, September 1992, IEEE

[6] Raj E., Tempero E., Levy H., Black A., Hutchinson N., Jul E.: A General-Purpose Programming Language. Software-Practice and Experience, 21 (1), pp. 91–118, January 1991

[7] Tenenbaum A.S., van Renesse R.: Distributed operating systems. Computing Surveys, 17 (4), pp. 419–470, December 1985

[8] Wegner P.: Dimensions of Object-Based Language Design. In: Proceedings of the Second ACM Conference on Object-Oriented Programming Systems, Languages and Applications, pp. 168–182, October 1987.

Informujemy, że począwszy od marca br. wpłaty na prenumeratę INFORMATYKI są przyjmowane nie tylko przez Zakład Kolportażu Wydawnictwa SIGMA-NOT, lecz także przez wszystkie urzędy pocztowe nadawczo-odbiorcze oraz doręczycieli na terenie całego kraju.

KRZYSZTOF M. SACHA
Instytut Automatyki
Politechnika Warszawska

Projektowanie oprogramowania systemów wbudowanych

Oprogramowania systemów wbudowanych (ang. *embedded system*) odznacza się dwoma cechami, istotnie wpływającymi na proces projektowania:
- opracowanie programów wymaga współpracy wieloosobowego zespołu,
- utrzymanie programu w eksploatacji wymaga jego okresowej konserwacji, tzn. modyfikacji, przeprowadzanej przez zmieniające się zespoły ludzi.

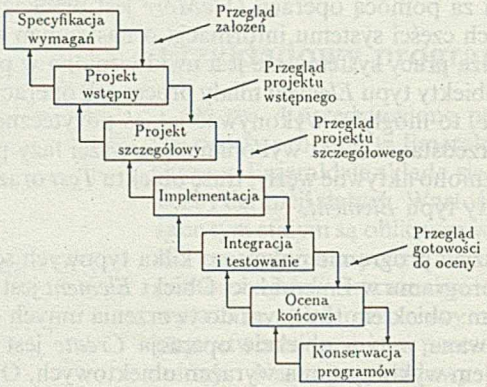
Celem wszystkich metod projektowych jest zmniejszenie kosztów opracowania i konserwacji oraz przyspieszenie tych działań. Równie istotne jest zagwarantowanie dostatecznie wysokiej niezawodności programów. Realizacja tych celów wymaga ścisłego dokumentowania struktury i sposobu działania programów oraz efektywnego przekazywania informacji o programie pomiędzy członkami zmieniających się zespołów projektowych.

Cykl opracowania oprogramowania

Punktem wyjścia wszystkich prac projektowych są założenia wstępne (ang. *preliminary requirements statement*), określające cele i funkcje oprogramowania. Założenia wstępne mogą być dostarczone przez użytkownika, jednak częściej są wynikiem projektu systemu i stanowią część jego specyfikacji.

Cykl opracowania oprogramowania (ang. *software life-cycle*) można opisać za pomocą klasycznego modelu „wodospadu" (rys. 1 – *waterfall model*), tworzącego sekwencję następujących po sobie kroków. W każdym kroku powstaje obszerna dokumentacja, której opracowanie i zatwierdzenie jest uważane za kryterium zakończenia danego kroku [6].

Analiza i specyfikacja wymagań (ang. *requirements analysis and specification*)
W pierwszym kroku są analizowane i dokumentowane potrzeby i wymagania użytkownika. Jednocześnie określa się ograniczenia narzucone na środowisko programowe i wykonawcze (typy komputerów, systemy operacyjne, języki programowania) oraz kryteria akceptacji gotowego produktu. Podstawowym wynikiem tego kroku jest dokument założenia oprogramowania (ang. *software requirements specification*), określający funkcje wykonywane przez oprogramowanie, sprzęgi z otoczeniem oraz dokładność, szybkość i wydajność obliczeń.



Rys. 1. Model cyklu opracowania oprogramowania

Projekt wstępny (ang. *preliminary design*)
Celem tego kroku jest podział programu na podstawowe komponenty, przypisanie funkcji wymienionych w założeniach do poszczególnych komponentów, zdefiniowanie sposobu współpracy komponentów oraz określenie sposobu ich testowania. Podstawowymi wynikami kroku są dwa dokumenty:
- wstępny opis projektu (ang. *preliminary design description*), opisujący podział oprogramowania na komponenty i określający funkcje poszczególnych komponentów,
- plan testów (ang. *test plan*), określający sposób testowania oraz oceny produktu.

Projekt szczegółowy (ang. *detailed design*)
W trzecim kroku następuje dalszy podział komponentów aż do poziomu jednostek programowych oraz ustalenie wspólnych struktur danych i algorytmów. W ostatniej fazie ustala się fizyczną strukturę programów, tzn. ich podział na jednostki kompilacyjne (ang. *program packaging*). Wynikiem projektu są dokumenty:
- opis projektu (ang. *detailed design description*), opisujący strukturę oprogramowania, funkcje i algorytmy działania wszystkich jednostek programowych oraz podstawowe struktury danych,
- opis testów (ang. *software test description*), definiujący wszystkie testy w terminach danych wejściowych, oczekiwanych rezultatów i kryteriów oceny.

Programowanie i testowanie jednostek programowych (ang. *implementation*)
Wykonanie tego kroku obejmuje napisanie i uruchomienie

Dr KRZYSZTOF SACHA ukończył studia na Wydziale Elektroniki Politechniki Warszawskiej w 1973 r., a w 1976 r. obronił pracę doktorską z zakresu informatyki. Obecnie pracuje w Instytucie Automatyki PW, zajmując się problemami projektowania oprogramowania systemów czasu rzeczywistego.

programów wszystkich jednostek oraz przygotowanie programów testujących. Wynikiem wykonania kroku są trzy dokumenty:
- kod źródłowy (ang. *source code*) oraz wydruk programu,
- instrukcje i raporty testowania modułów,
- instrukcje testowania (ang. *test procedures*), tzn. programy testujące z instrukcjami uruchomienia, wykonania i sposobu oceny rezultatów.

Integracja i testowanie komponentów (ang. *integration and testing*)

W kolejnym kroku następuje łączenie jednostek programowych w komponenty coraz wyższego poziomu, aż do poziomu całego programu. Wszystkie komponenty są testowane zgodnie z opracowanymi instrukcjami testowania. Wyniki wykonania kroku zostają zebrane w postaci dokumentów:
- uaktualniona specyfikacja projektu i źródłowy kod programu,
- raporty testowania komponentów.

Instalacja i ocena końcowa (ang. *installation and validation*)

Ostatnim krokiem opracowania oprogramowania jest jego instalacja w docelowym środowisku wykonawczym oraz ocena przeprowadzana przez ostateczne testowanie całego oprogramowania według opracowanej instrukcji testowania. Krok oceny powinien być wykonany przez specjalny zespół testujący, niezależny od zespołu opracowującego oprogramowanie. Wynikiem kroku są dokumenty:
- raporty testowania oprogramowania,
- uaktualniony opis projektu oraz źródłowy kod programu.

Dokumenty projektowe podlegają ocenie (np. rad technicznych) podczas przeglądów odbywanych po zakończeniu kolejnych kroków (rys. 1). Kryteria oceny obejmują: czytelność, jednoznaczność, spójność wewnętrzną oraz spójność i powiązanie odnośnikami całego zbioru dokumentów.

Konserwacja oprogramowania (ang. *maintenance*).

Ocena końcowa i wprowadzenie do eksploatacji nie kończą prac nad oprogramowaniem. Programy sterujące są z reguły wykorzystywane przez wiele lat, podczas których zarówno sprzęt komputerowy jak i oprzyrządowanie procesu sterowanego ulegają zmianom i modyfikacjom. Zmieniają się też wymagania użytkowników. W każdym z tych przypadków odpowiednim zmianom musi ulec oprogramowanie. Powodem zmian mogą być też wykryte błędy. Faza konserwacji rozciąga się aż do wycofania programów z eksploatacji.

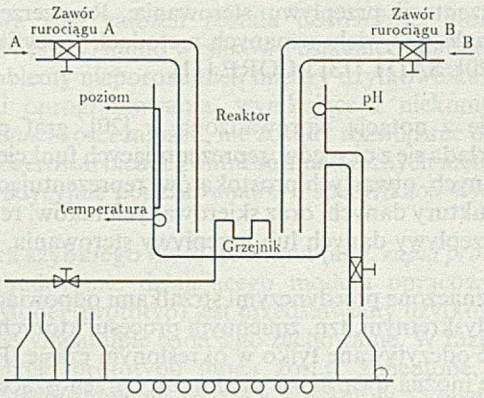
Koszty związane z konserwacją oprogramowania przewyższają na ogół koszty oryginalnego projektu. Przyjmując koszt oryginalnego projektu (od specyfikacji wymagań do instalacji i oceny) za 100%, można oszacować podział kosztów następująco:

● specyfikacja wymagań	15%
● projekt wstępny	7%
● projekt szczegółowy	18%
● programowanie modułów	20%
● integracja i testowanie komponentów	20%
● instalacja i ocena	20%
łącznie	100%
● konserwacja	70...200%

Przykład

Przegląd metod specyfikacji i projektowania oprogramowania podany w dalszej części artykułu będzie ilustrowany przykładem systemu sterującego instalacją chemiczną współpracującą z liniami butelkowania produktu. Opis instalacji został zaczerpnięty z [20].

Centralnym elementem instalacji (rys. 2) jest reaktor przepływowy, do którego są doprowadzane rurociągami stężone roztwory dwóch składników *A* i *B*, dozowanych w taki sposób, aby utrzymać stałą kwasowość (pH) produktu. Stężenie składników nie jest stałe i waha się przypadkowo w określonych granicach. Proporcja dozowania składników zależy od chwilowego stężenia, jednak przeciętnie składnik *A* stanowi ok. 90%, a *B* – ok. 10% masy reakcji. Wzrost temperatury zwiększa wydajność procesu, może jednak grozić eksplozją w razie przegrzania.



Rys. 2. Instalacja chemiczna

Ciekły produkt reakcji jest butelkowany przez kilka automatycznych linii, z których każda składa się z magazynu butelek, podajnika pozycjonującego pustą butelkę na platformie wagi, czujnika obecności butelki na platformie i zaworu otwierającego dopływ produktu do butelki.

System sterujący powinien regulować poziom, kwasowość i temperaturę cieczy w reaktorze, sterować pracą wszystkich linii butelkowania oraz obrazować stan instalacji dla dyspozytora procesu.

Tradycyjne metody projektowania oprogramowania

Większość metod analizy i specyfikacji wymagań buduje mniej lub bardziej sformalizowany model koncepcyjny projektowanego systemu. Model podlega dekompozycji przez określenie takiej struktury systemu, która w jak największym stopniu ułatwi zrozumienie zadań, które mają być wykonane. Efektywność przyszłej implementacji nie jest brana pod uwagę.

Analiza i specyfikacja wymagań

Najważniejszym wyróżnikiem metody jest przyjęte kryterium dekompozycji problemu. Najstarsze metody dekompozycji funkcjonalnej [5] traktują funkcje oprogramowania jako funkcję złożoną (w sensie matematycznym), dekomponowaną na funkcje składowe. Problemem metody jest brak wskazówek co do sposobu dekompozycji. W rezultacie projekty prowadzone zgodnie z zasadą dekompozycji funkcjonalnej cechują się niepowtarzalnością wyniku, a projektowanie opiera się bardziej na intuicji niż wiedzy.

Późniejsze metody analiży strukturalnej zalecają prowadzenie dekompozycji tak, aby dopasować strukturę opisu do:
- struktury przepływu danych między różnymi funkcjami systemu, lub

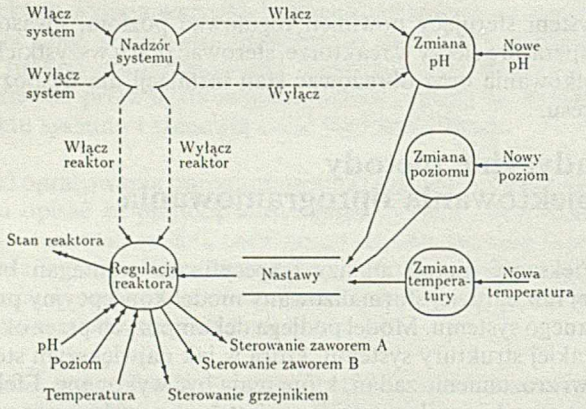
● przepływu sterowania niezbędnego dla zrealizowania wymaganego przetwarzania danych wejściowych w wyjściowe.

Najczęściej używanym narzędziem dekompozycji są grafy przepływu danych (ang. *data flow graph*), szczególnie często używane do opisu przetwarzania w systemach administracyjnych, bankowych lub komercyjnych. Przykład wykorzystania grafów przepływu do specyfikacji i projektowania systemu przetwarzania danych metodą Gane-Sarsona można znaleźć w [18].

Wykorzystanie grafów przepływu danych do opisu systemów wbudowanych wymaga rozszerzenia przez wprowadzenie jawnej reprezentacji przepływu sterowania. Rozszerzenia takie można znaleźć w wielu uznanych metodach specyfikacji, np.: SA/SD [20], SADT [13] i CORE [11].

Zgodnie z notacją wprowadzoną w [20], graf przepływu danych składa się z okręgów, reprezentujących funkcje przetwarzania danych, otwartych prostokątów, reprezentujących zbiory lub struktury danych, oraz skierowanych łuków, reprezentujących przepływy danych lub przepływy sterowania.

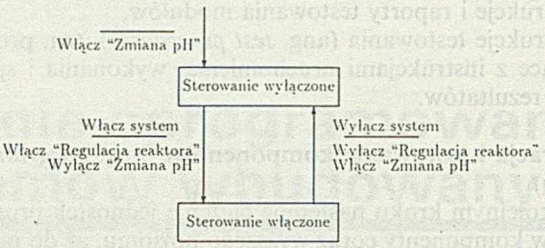
Łuki oznaczone pojedynczymi strzałkami odpowiadają przepływom dyskretnym, tzn. zmiennym procesu, których wartości mogą być odczytywane tylko w określonym czasie. Przepływy dyskretne można uważać za wiadomości przenoszące wartości zmiennych i powodujące wykonanie funkcji, do których są skierowane. Przepływy dyskretne mogą również powodować zapamiętywanie danych w zbiorach. Np. na rys. 3 funkcje *Zmiana pH*, *Zmiana poziomu* i *Zmiana temperatury* są wykonywane w chwili pojawienia się zewnętrznych przepływów przenoszących nowe wartości nastaw. Wykonanie każdej z tych funkcji powoduje zapamiętanie nowej wartości parametru w zbiorze *Nastawy*.



Rys. 3. Graf przepływu danych w systemie sterującym reaktorem chemicznym

Łuki wyróżnione podwójnymi strzałkami oznaczają ciągłe przepływy danych, których wartości są zawsze dostępne i mogą być odczytane w dowolnym momencie. Zawsze dostępne są również dane zapisane w zbiorach danych, oznaczanych na rysunku jako niedomknięte prostokąty. Funkcja *Regulacja reaktora* rejestruje wartości ciągłych zmiennych *pH*, *Poziom*, i *Temperatura*, odczytuje nastawy ze zbioru danych *Nastawy* i aktualizuje wartości zmiennych sterujących *Sterowanie zaworem A*, *Sterowanie zaworem B* i *Sterowanie grzejnikiem*. Wartości zmiennych ciągłych są – przynajmniej w teorii – ustalane i zmieniane w sposób ciągły. Szczegóły realizacyjne są na poziomie założeń nieistotne.

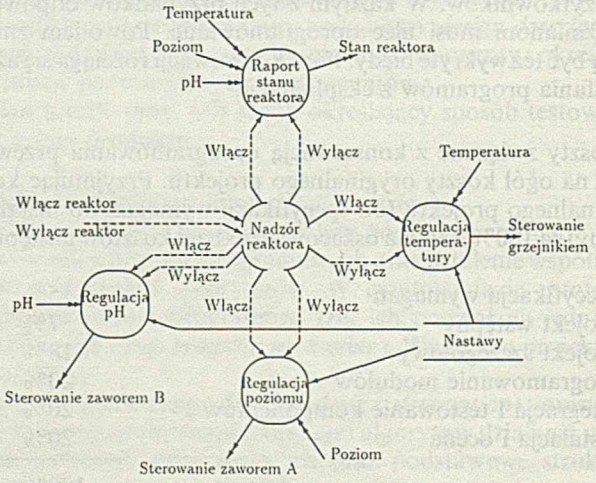
Przepływ sterowania jest reprezentowany w grafie przez funkcję sterującą *Nadzór reaktora* oraz łuki sterujące, rysowane jako przerywane linie zakończone pojedynczymi strzałkami. Funkcja sterująca jest definiowana jako automat skończony, opisany przez odpowiedni graf przejść (rys. 4).



Rys. 4. Graf przejść funkcji „Nadzór reaktora"

Graf przepływu danych z rys. 3 definiuje wymagania bardziej precyzyjnie niż podany na początku oryginalny opis problemu. Np. problem relacji między funkcjami regulacji stanu reaktora i zmiany bieżących nastaw nie został dostrzeżony w oryginalnym opisie zadania, chociaż istnieje w praktyce. W przedstawionym przykładzie przyjęto, że nastawy dotyczące poziomu i temperatury cieczy w reaktorze, które wpływają tylko na wydajność procesu, a nie na jakość produktu, mogą być zmieniane na bieżąco podczas pracy instalacji. Natomiast zmiana wartości zadanej pH, która zmienia jakość produktu, może być dokonana tylko przy zatrzymanym biegu procesu.

Każdą z funkcji przedstawionych na rys. 3 można zdekomponować na funkcje prostsze, połączone przez występujące między nimi przepływy. Jednak liczba szczegółów, jakie można przedstawić na pojedynczym diagramie (grafie) jest ograniczona. Dalsze zwiększanie szczegółowości, tzn. liczby węzłów grafu i liczby łączących je łuków, może uczynić rysunek nieczytelnym. Alternatywną metodą jest zastosowanie reprezentacji hierarchicznej, w której dekompozycja poszczególnych funkcji z diagramu wyższego poziomu jest przedstawiana na osobnych rysunkach, tworzących kolejny poziom opisu. Warunek spójności dekompozycji wymaga zgodności przepływów wejściowych i wyjściowych dekomponowanej funkcji z przepływami wchodzącymi i wychodzącymi do i z grafu niższego poziomu. Przykład dekompozycji funkcji *Regulacja reaktora* przedstawiono na rys. 5.



Rys. 5. Szczegółowy diagram funkcji „Regulacja reaktora"

Rozwijanie hierarchii diagramów przepływu danych jest kontynuowane aż do poziomu szczegółowości, na którym wszystkie funkcje grafu mogą zostać opisane w postaci np. zdań pseudokodu lub formuł (równań lub funkcji) matematycznych.

Hierarchia diagramów, wraz z opisem funkcji elementarnych oraz opisem danych gromadzonych w zbiorach i przepływających między funkcjami, definiuje w sposób kompletny i jednoznaczny pożądane zachowanie projektowanego systemu.

Projektowanie programu

Metody stosowane w fazie projektowania uzupełniają model koncepcyjny, opracowany podczas specyfikacji wymagań, o szczegóły dotyczące technologii realizacyjnej oraz przekształcają jego strukturę tak, aby osiągnąć wymaganą wydajność oprogramowania. W tym celu poszczególne funkcje są przypisywane do określonych komponentów jednostek programowych. Rezultatem projektu jest *model implementacyjny*, opisujący dokładnie sposób wykonania wszystkich funkcji.

Powróćmy do systemu sterującego reaktorem chemicznym. Problemy, które muszą być rozważone podczas projektowania programu, to np.: *Czy sterowanie reaktorem ma realizować system rozproszony, program sekwencyjny, czy zespół współbieżnych zadań?*, lub w systemie wielozadaniowym: *Czy poziom, temperatura i pH powinny być regulowane przez trzy różne zadania, czy przez jedno zadanie regulacji PID?*. Podczas analizy wymagań obydwa pytania były nieistotne, ale podczas wykonywania projektu stanowią istotny problem, gdyż różne odpowiedzi prowadzą do różnych implementacji.

Faza projektowania powinna być niezależna od fazy specyfikacji. Praktyka wykazuje jednak, że bardzo trudno jest przełamać schemat dekompozycji systemu narzucony przez przyjętą metodę specyfikacji. Stąd, metody projektowania są z reguły związane z określonymi metodami specyfikacji w tym sensie, że używają podobnego modelu formalnego i podobnej notacji, jak np. CORE/MASCOT. Niektóre metody, jak np. SA/SD lub SADT, obejmują obydwie fazy cyklu projektowego.

Zależnie od wybranego modelu dekompozycji, przypisanie funkcji określonych w założeniach do komponentów programu może być dokonane ze względu na różne kryteria. Metody projektowania oparte na diagramach przepływu danych (np. SA/SD, SADT i MASCOT [17]) dążą do minimalizacji przepływów między komponentami.

Innym kryterium jest dopasowanie struktury programu do struktury danych opisujących rozwiązywany problem (np. JSP [9]). Zwolennicy tego podejścia argumentują, że struktury danych stanowią najbardziej stabilny model rzeczywistości, w której działa program, znacznie trwalszy od modelu funkcji, które mogą ulegać zmianie wraz ze zmianami wymagań użytkownika. W każdym przypadku, nieodłączną częścią modelu implementacyjnego jest jawne pokazanie przepływu sterowania.

Celem wszystkich metod projektowania jest osiągnięcie zadanej funkcjonalności i wydajności oprogramowania oraz określenie jego struktury w sposób jak najbardziej czytelny i zrozumiały. Czytelność struktury w istotny sposób ułatwia testowanie i modyfikowanie gotowego programu.

Notacje graficzne nie zastępują, lecz uzupełniają tekst założeń i specyfikację projektu. Podstawowe treści, które nie mogą być na ogół wyrażone na diagramach obejmują:

- szczegółowe opisy elementarnych funkcji występujących na diagramach,
- opis wymagań niefunkcjonalnych, dotyczących żądanej szybkości, dokładności i wydajności oprogramowania,
- analizy i uzasadnienia podjęcia takich, a nie innych decyzji projektowych.

Szybkie makietowanie

Jakość założeń oprogramowania, a szczególnie prawidłowe rozpoznanie wymagań użytkownika i jednoznaczne ich udokumentowanie mają podstawowe znaczenie dla powodzenia projektu programowego. Błędy popełnione w założeniach pozostają na ogół niewidoczne aż do ostatniego kroku instalacji i oceny programów. Z kolei korekcja błędów założeń wykrytych dopiero po implementacji wymaga często projektowania struktury programów, poprawienia ich kodu i powtórnego przetestowania wszystkich zmienionych modułów i sprzęgów między modułami.

Kluczowa rola założeń została dostrzeżona stosunkowo wcześnie. Niestety, mimo wysiłków i osiągnięć analizy strukturalnej, problemy nieporozumień między użytkownikami i projektantami oprogramowania, wynikające z niekompletności i niejednoznaczności założeń, nie zostały rozwiązane do końca. Ten fakt sugerował nieadekwatność analitycznych metod stosowanych tradycyjnie podczas analizy i specyfikacji wymagań.

Technika **szybkiego makietowania** (ang. *rapid prototyping*) zakłada zbudowanie działającego modelu opracowywanego systemu. Model (prototyp) jest przedstawiany użytkownikowi, który eksperymentalnie bada jego zachowanie. W razie wątpliwości, funkcje prototypu mogą zostać zmienione, co daje możliwość wypróbowania różnych wariantów. Główne różnice między prototypem, a finalnym produktem mogą dotyczyć dwóch zagadnień:

- prototyp może nie osiągać wymaganej wydajności (dokładności, szybkości, liczby końcówek, itp.) i ograniczać się do modelowania funkcji,
- prototyp może być niekompletny i wykonywać tylko niektóre wymagane funkcje, np. dialogu z operatorem.

Technika szybkiego makietowania nie narusza tradycyjnego modelu opracowania oprogramowania. Prototyp programu jest budowany i oceniany w kroku analizy i specyfikacji wymagań. Po zatwierdzeniu, prototyp może stać się częścią założeń oprogramowania. Główną korzyścią wynikającą z posiadania prototypu jest możliwość lepszego rozpoznania rzeczywistych potrzeb użytkownika oraz bardziej wiarygodna ocena przewidywanego zachowania programu i jego współdziałania ze światem zewnętrznym. Bardzo często pozwala to opóźnić i ograniczyć zakres konserwacji.

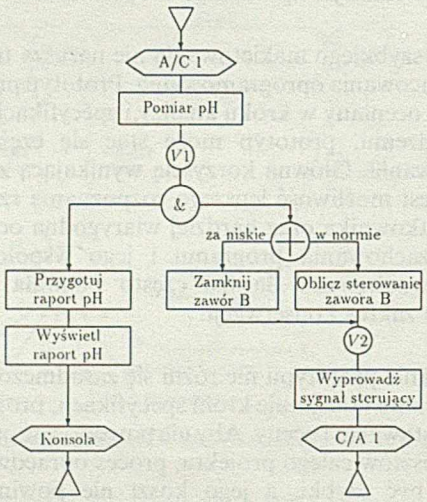
Opracowanie prototypu nie różni się zasadniczo od opracowania programu i obejmuje kroki specyfikacji, projektu, implementacji, testowania i oceny. Aby nie powodować opóźnień i nie podnosić kosztów całego projektu, proces opracowania prototypu musi być szybki, a jego koszt nie powinien istotnie przekraczać typowych kosztów kroku specyfikacji wymagań. Można to osiągnąć pod warunkiem używania odpowiednich narzędzi programowych, takich jak specjalizowane języki makietowania i generatory aplikacji. W wielu zastosowaniach komercyjnych, wykorzystujących tylko niewielki procent możliwości sprzętu komputerowego, prototyp może spełnić wszystkie wymagania dotyczące tak funkcji, jak wydajności, stając się w ten sposób implementacją.

Makietowanie oprogramowania czasu rzeczywistego jest trudniejsze, gdyż wymagania dotyczące funkcji i wydajności nie mogą być rozdzielone. Prowadzi to do koncepcji **wykonalnej specyfikacji** (ang. *executable specification*), która opisuje wszystkie wymagania, a jednocześnie stanowi prototyp badany i oceniany podczas symulacyjnego wykonania.

Przykładem dobrze określonej metody makietowania może być metoda SREM (*Software Requirements Engineering Method*) [1], oparta na dekompozycji systemu ze względu na wiadomości wejściowe. Sposób przetwarzania wiadomości opisuje się w postaci sekwencyjno-równoległej sieci działań, nazywanej R-siecią[1)].

Każda R-sieć określa sposób przetwarzania pojedynczej wiadomości wejściowej. Opis całego programu ma postać zespołu R-sieci, opisujących przetwarzanie wszystkich wiadomości wejściowych. Wewnątrz każdej R-sieci można określić punkty kontrolne (ang. *validation point*), w których symulator może rejestrować podczas wykonania czas przejścia przez punkt oraz wartości określonych danych. Wymagania niefunkcjonalne określa się przez podanie czasu oraz dokładności danych zbieranych w punktach kontrolnych.

Metoda SREM została opracowana dla dyskretnych systemów dowodzenia i sterowania, ale może być również stosowana do opisu systemów sterowania ciągłego, takich jak system sterujący reaktorem chemicznym. Przepływy wejściowe *pH*, *Poziom* i *Temperatura* odpowiadają wiadomościom wejściowym, interpretowanym jako kolejne, dyskretne pomiary. R-sieć, opisująca przetwarzanie wiadomości wejściowej *pH*, jest pokazana na rys. 6. Wiadomość ta jest wprowadzona przez sprzęt *A/C* 1. Przetwarzanie rozdziela się na dwie równoległe gałęzie. W jednej gałęzi jest opracowywany raport o wartości pH, wyświetlany przez sprzęg *Konsola*. W drugiej, zależnie od wartości pH, jest wykonywany albo algorytm obliczenia sygnału sterującego, albo zawór jest awaryjnie zamykany. W obydwu przypadkach sygnał sterujący jest wyprowadzany przez sprzęt *C/A* 1. Dokładność i punktualność obliczeń mogą być kontrolowane przez rejestrację wartości zmiennych oraz czasu w punktach kontrolnych *V*1 i *V*2.



Rys. 6. R-sieć wiadomości „wartość pH"

Przykład z rys. 6 pokazuje wyraźną różnicę między prototypem oprogramowania, a jego projektem i implementacją. Każda R-sieć może zostać zaprogramowana i wykonana, stanowiąc prototyp określonej funkcji. Z drugiej strony, jakkolwiek zespół R-sieci opisuje wymagania dotyczące przetwarzania danych wejściowych, to nie określa żadnej praktycznej struktury oprogramowania i nie może być utożsamiony z projektem.

[1)] R-net od Response-net, sieć określająca odpowiedź systemu na pobudzenie wejściowe

Technika makietowania może być również użyteczna w fazie projektowania oprogramowania. Opracowanie prototypu oprogramowania o strukturze określonej podczas projektu (wstępnego lub szczegółowego) umożliwia eksperymentalną ocenę podjętych decyzji. W razie niezadowalających rezultatów, prototyp może zostać zmodyfikowany, co pozwala na wiarygodne sprawdzenie różnych wariantów projektu.

Projektowanie obiektowe

Celem fazy projektowania oprogramowania (zarówno projektu wstępnego jak i szczegółowego) jest określenie struktury programu. Należy przy tym odróżnić strukturę logiczną, tzn. wyodrębnienie współbieżnych zadań, podział kodu programu na podprogramy i określenie reprezentacji struktur danych, oraz fizyczną, określającą upakowanie zadań, podprogramów i struktur danych w fizycznie wyodrębnionych modułach (pakietach). Właściwa, tzn. czytelna, efektywna i modyfikowalna struktura (logiczna i fizyczna) ułatwia zarówno implementację i testowanie programów, jak i ich konserwację. W ostatecznym rezultacie prowadzi to do zmniejszenia kosztów najbardziej kosztownych kroków cyklu życia oprogramowania.

Tradycyjne metody projektowania oprogramowania opóźniają moment podjęcia decyzji o fizycznej strukturze programu do ostatniego etapu projektu. W pierwszym etapie zostaje określana struktura logiczna, przy czym dominującym podejściem jest stopniowa dekompozycja systemu na coraz prostsze komponenty. W każdym kroku dekompozycji określa się kompletny opis systemu z rosnącą dokładnością. W ostatnim etapie (pakietyzacji) poszczególne jednostki programowe są łączone w moduły (pakiety, zbiory), stanowiące samodzielne jednostki kompilacyjne.

Podstawowym celem pakietyzacji jest ułatwienie dokumentowania i modyfikowania programu. Pakietyzacja powinna być wykonana tak, aby ewentualne zmiany, dokonane w jednym module, nie powodowały konieczności wprowadzania zmian do innych modułów. Niestety, stwierdzenie to jest na ogół niekonstruktywne, tzn. nie daje wskazówek, w jaki sposób grupować jednostki programowe w moduły.

W praktyce, kryteria stosowane podczas pakietyzacji programu są różne. Metody oparte na modelach grafów przepływu danych dążą z reguły do budowy pakietów o silnych powiązaniach wewnętrznych oraz jak najmniejszych przepływach międzymodułowych. Innym stosowanym kryterium jest grupowanie w jednym module funkcji o podobnym charakterze (np. pakiet procedur graficznych). Metody oparte na modelu struktur danych budują moduły zawierające operacje na poszczególnych strukturach.

Metody **projektowania obiektowego** (ang. *object-oriented design*) opierają się na odmiennych przesłankach. Punktem wyjścia jest spostrzeżenie, że struktura oprogramowania powinna odwzorować strukturę otoczenia (realnego świata), w którym jest zdefiniowany problem. Naturalną reprezentację otoczenia stanowi zbiór obiektów, opisywanych przez odpowiednie struktury danych i operacje wykonywane na obiektach przez inne obiekty. Struktura danych wraz z zestawem operacji, które można wykonywać na tych danych, tworzy abstrakcyjny typ danych. Metody projektowania obiektowego opierają się na koncepcjach strukturalizacji programu wokół struktur danych oraz ukrywania nieistotnych szczegółów implementacji wewnątrz definicji typów danych. Poszczególne moduły programu odpowiadają obiektom lub klasom obiektów wyodrębnionych w opisie problemu.

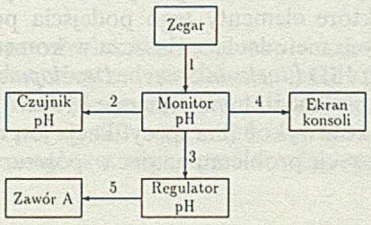
Metoda projektowania obiektowego nie narusza cyklu opracowania oprogramowania, przedstawionego na rys. 1. Przebieg projektu można opisać w postaci sekwencji kilku kroków [2]:
1) określenie zbioru obiektów,
2) określenie zbioru operacji wykonywanych na obiektach i wymaganych przez obiekty,
3) określenie sprzęgu między obiektami,
4) opracowanie projektu oraz implementacja poszczególnych operacji.

Podział na obiekty o ściśle zdefiniowanych operacjach (kroki 1...3) określa fizyczną strukturę programu. Struktura logiczna powstaje przez stopniową dekompozycję funkcji obiektów dokonywaną w ostatnim kroku (4). W porównaniu z metodami tradycyjnymi następuje tu odwrócenie kolejności pakietyzacji programu i dekompozycji jego funkcji.

Obiekty i operacje w regulatorze pH

Obiekt	Operacja	
	wykonana przez obiekt	wykonana na obiekcie
Zegar	Start cyklu (1)	
Monitor pH	Wykonaj pomiar (2) Przekaż wartość (3) Wyświetl wartość (4)	Start cyklu (1)
Czujnik pH		Wykonaj pomiar (2)
Regulator pH	Ustaw zawór *A* (5)	Przekaż wartość (3)
Ekran konsoli		Wyświetl wartość (4)
Zawór A		Ustaw zawór *A* (5)

Powróćmy dla przykładu do regulatora pH przedstawionego na rys. 6. Można tu wyróżnić następujące obiekty fizyczne: czujnik pH, regulator, ekran (konsoli), zawór *A* oraz zegar, wymuszający cykliczne działanie regulatora. Program realizujący funkcję sterowania pH można podzielić na analogiczne obiekty programowe. Listę operacji wykonywanych przez obiekty na obiektach zebrano w tabeli. Współdziałanie obiektów przedstawia graficznie diagram pokazany na rys. 7. Warto zauważyć, że widoczne na diagramie łuki nie oznaczają przepływów danych między operacjami, lecz kierunek wykonania operacji: od obiektu wykonującego operację do obiektu, na którym operacja jest wykonana.



Rys. 7. Diagram dekompozycji obiektowej regulatora pH

Implementacja projektu zależy od środowiska programowego. W systemie wielozadaniowym poszczególne obiekty mogą być zaprogramowane jako współbieżne, współpracujące ze sobą zadania. Właściwa synchronizacja operacji wykonywanych przez poszczególne obiekty jest zapewniana przez odpowiednią synchronizację zadań.

W sekwencyjnym systemie jednoprogramowym każdy obiekt stanowi moduł programu (np. pakiet Ady), złożony ze struktur danych opisujących stan obiektu i zespołu podprogramów zmieniających ten stan. Przepływ sterowania między obiektami reguluje program główny, realizujący wymagane funkcje przez wywoływanie operacji na poszczególnych obiektach.

Implementacja oprogramowania o strukturze obiektowej wymaga języków programowania wyposażonych w mechanizmy definiowania obiektów, np. Smalltalk oraz C++. Metoda projektowania obiektowego jest również powszechnie stosowana przy opracowywaniu programów w języku Ada [3]. Obiekty przyjmują tu postać pakietów, tzn. jednostek programowych obejmujących dane oraz operacje na danych. Co więcej, pakiety rodzajowe pozwalają na definiowanie wzorców, zgodnie z którymi mogą być powoływane konkretne pakiety, różniące się wprawdzie szczegółami (parametrami), lecz dziedziczące zasadnicze własności strukturalne.

Zalety projektowania obiektowego w pełni uwidaczniają się w fazie konserwacji. Sieć współpracujących obiektów stanowi naturalny model problemu, który jest najbardziej stabilną częścią projektu oprogramowania. Wszystkie istotne cechy obiektów rzeczywistych są zgrupowane w odpowiadających im obiektach programowych, stąd każda zmiana w otoczeniu programu prowadzi jedynie do lokalnych zmian wewnątrz określonych pakietów programu. Co więcej, wszystkie istotne dane, opisujące poszczególne obiekty, są również zgrupowane (ang. *encapsulated*) wewnątrz poszczególnych modułów, dzięki czemu wpływ ewentualnych zmian reprezentacji danych nie przekracza granic pakietów.

Transformacyjna implementacja

Najbardziej twórczą częścią projektu programowego jest analiza i specyfikacja wymagań. Powstające w tym kroku założenia (specyfikacja wymagań) dokładnie opisują pożądane zachowanie systemu. Wszystkie działania podejmowane w dalszych krokach projektu zmierzają do implementacji tego zachowania, nie mogą natomiast zmieniać żadnych elementów założeń. W istocie więc założenia w pełni opisują semantykę projektowanego systemu. Dalsze zmiany mogą dotyczyć jedynie składni tego opisu.

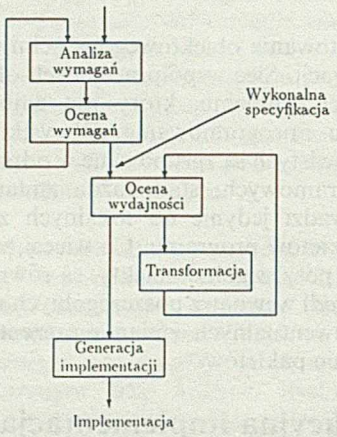
To spostrzeżenie prowadzi do koncepcji **transformacyjnej implementacji**, zgodnie z którą formalnie zapisana specyfikacja (założenia) jest automatycznie lub półautomatycznie przekształcana w implementację [21]. Zapisana formalnie specyfikacja jest wykonalna i stanowi pierwszy prototyp oprogramowania. Po ocenie i zatwierdzeniu zostaje ona poddana transformacjom, które zmieniają strukturę prototypu tak, aby osiągnąć żądaną wydajność, bez zmieniania ustalonej funkcjonalności.

Stosowane transformacje mogą być wybierane automatycznie lub ręcznie, natomiast ich wykonanie powinno przebiegać zawsze automatycznie. Jeżeli będą stosowane wyłącznie transformacje o udowodnionej poprawności, tzn. nie zmieniające zachowania, to proces testowania implementacji stanie się zbędny. Na każdym etapie transformacji aktualna specyfikacja może być traktowana jako kolejny prototyp oprogramowania, którego funkcje, wydajność i koszt mogą być badane eksperymentalnie.

Potencjalne zalety tego podejścia dotyczą całego cyklu opracowania oprogramowania. Przede wszystkim znikają pracochłonne etapy projektowania, implementacji i testowania. Bez zmian pozostaje krok specyfikacji wymagań, po którym następuje ocena prototypu, transformacja struktury i generacja kodu implementacji (rys. 8 [7]).

Jedyną fazą cyklu, która musi być wykonana ręcznie jest faza specyfikacji wymagań, której produkt (specyfikacja) stanowi podstawową dokumentację projektu. Podczas konserwacji wszelkim zmianom podlega również specyfikacja, a nie kod implementacji. Po wprowadzeniu zmian, zmodyfikowana specyfikacja jest ponownie transformowana do postaci implementacji.

Rozważmy ponownie system sterujący instalacją chemiczną. Funkcje systemu można rozbić na niezależne procesy:
- rejestrowania pH, poziomu i temperatury,
- zapamiętywania wartości zmierzonych,
- przetwarzania sytuacji awaryjnych (np. przekroczenie zakresu pH),
- regulacji pH, poziomu i temperatury,
- taktowania rytmu pracy.

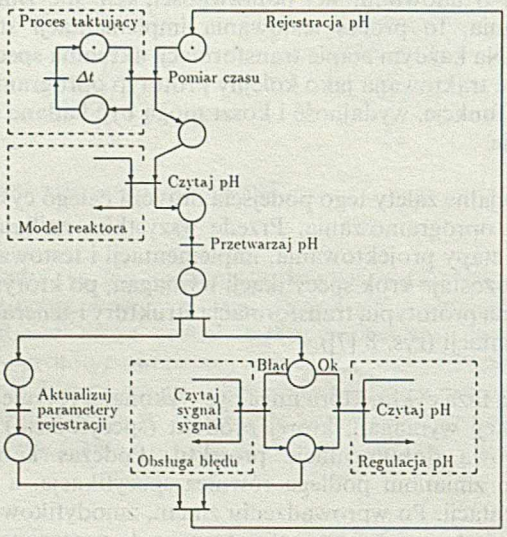


Rys. 8. Model transformacyjnej implementacji

Wyodrębnienie dużej liczby współbieżnych procesów pozwala uniknąć arbitralnego narzucania kolejności wykonywania działań i czyni specyfikację niezależną od implementacji. Każdy proces oraz jego połączenia z innymi procesami może zostać opisany w postaci rozszerzonej sieci Petriego.

Sieć Petriego [12] jest grafem skierowanym, zbudowanym z dwóch rodzajów wierzchołków: miejsc i tranzycji, połączonych łukami tak, że żadne dwa miejsca ani żadne dwie tranzycje nie są połączone bezpośrednio. Miejsca są przedstawiane graficznie jako okręgi, a tranzycje – jako krótkie kreski.

Miejsce może zawierać znacznik rysowany w postaci kropki. Znaczniki poruszają się między miejscami w rezultacie odpalania tranzycji. Tranzycja jest gotowa do odpalenia (wzbudzona), jeżeli wszystkie jej miejsca wejściowe zawierają znaczniki. Odpalenie tranzycji powoduje usunięcie znacznika z każdego z miejsc wejściowych i umieszczenie znacznika w każdym miejscu wyjściowym.



Rys. 9. Proces rejestracji pH z procesami współpracującymi

Rozszerzona sieć Petriego zawiera zmienne i funkcje [14, 15]. Znakowanie sieci, tzn. rozkład znaczników w miejscach, odpowiada stanom sterowania wykonaniem systemu modelowanego przez sieć. Tranzycje odpowiadają akcjom i są związane z funkcjami. W chwili odpalenia tranzycji jest obliczana związana funkcja i jej wartość zostaje przypisana odpowiednim zmiennym. Przykładową sieć procesu *Rejestracja pH* przedstawiono na rys. 9.

Miejsce, z którego wychodzi kilka łuków skierowanych do różnych tranzycji, modeluje konstrukcję warunkową. Jeżeli miejsce to zawiera znacznik, to odpalić może tylko jedna z jego tranzycji wyjściowych. Wybór tranzycji dokonuje się przez obliczenie warunków logicznych, przypisanych do poszczególnych łuków.

Specyfikacja opracowana w postaci rozszerzonej sieci Petriego jest wykonalna, a wykonywane przez nią funkcje mogą być ocenione podczas wykonania sieci (pod warunkiem opracowania programów poszczególnych funkcji elementarnych). W fazie transformacji struktura sieci może zostać zmieniona tak, aby spełnić ograniczenia środowiska wykonawczego:
- W systemie wielozadaniowym można połączyć procesy rejestracji pH, poziomu i temperatury w jedno zadanie rejestracji, odczytujące dane z przetworników analogowo-cyfrowych i wykonujące wymagane algorytmy filtracji i przeliczenia jednostek. Podobnie, procesy regulacji stanu można połączyć w jedno zadanie regulatora. Pozostałe procesy mogą zostać zaimplementowane przez odpowiednie funkcje systemu operacyjnego;
- W systemie rozproszonym można połączyć procesy rejestracji i regulacji pH w jedno zadanie obsługi pH, implementowane przez dedykowany mikrosterownik tego parametru. Podobnie mogą zostać przekształcone procesy związane z regulacją poziomu i temperatury. Komunikacja między procesami może zostać zrealizowana przez systemowe mechanizmy przekazywania wiadomości.

Efektywne wykorzystania w praktyce projektowej koncepcji transformacyjnej implementacji wymaga opracowania szczegółowych metod oraz oprogramowania narzędziowego, wspomagającego etapy makietowania, oceny i transformacji. Obecnie żaden z tych elementów nie jest jeszcze w pełni dostępny, natomiast niektóre elementy tego podejścia pojawiają się w opracowywanych metodach, zwłaszcza w komercyjnie dostępnych metodach JSD (*Jackson System Development*) [4] i TAGS [16] oraz w eksperymentalnym systemie Model [19]. W każdym z tych przypadków wykonalna specyfikacja jest opracowywana przez dekompozycję problemu na sieć współpracujących procesów.

Metody formalne

W fazie projektowania i implementacji, założenia (specyfikacja wymagań) są przekształcane na kod programu, napisany w wybranym języku programowania. Weryfikacja stwierdzająca, czy otrzymany kod poprawnie realizuje wymagane funkcje jest celem długotrwałego procesu testowania, który pochłania ok. 40% kosztów oryginalnego projektu. Co więcej, proces testowania nie daje gwarancji całkowitej poprawności programu.

Odmienne podejście do problemu poprawności programów opiera się na formalnych (matematycznych) metodach dowodzenia poprawności [8]. Ponieważ poprawność programu bada się względem specyfikacji, metody te wymagają zapisania specyfikacji w postaci sformalizowanych formuł matematycznych. Właściwości programu są formułowane w postaci twierdzeń, które mogą być dowodzone opierając się na założeniach

zawartych w specyfikacji. Podstawową trudność w stosowaniu tych metod stanowi fakt, że przeprowadzenie dowodu poprawności jest na ogół trudniejsze od opracowania programu i wymaga dużego doświadczenia matematycznego.

Większe zastosowanie praktyczne znalazło nieco inne podejście, w którym konstrukcja dowodu poprawności programu przebiega równolegle z konstrukcją samego programu. Najbardziej znanym przedstawicielem tej grupy metod jest metoda VDM (*Vienna Development Method*) [10].

Opracowanie programu metodą VDM rozpoczyna się od formalnej specyfikacji wymagań, a kończy na gotowym kodzie programu. Specyfikacja opisuje działanie programu w terminach abstrakcyjnych typów danych, które definiują stan programu i operacji manipulujących danymi. Zarówno dane, jak i operacje, otrzymuje się z nieformalnego opisu problemu wyrażającego intencje użytkownika. Zgodność formalnej specyfikacji z nieformalnymi wymaganiami nie może być sprawdzona formalnie.

Dopuszczalne stany programu charakteryzuje się za pomocą predykatów nazywanych niezmiennikami typów danych. Niezmienniki muszą być spełnione we wszystkich stanach między wykonaniem operacji. Każda operacja jest charakteryzowana przez dwa inne predykaty:
- warunek początkowy (ang. *precondition*), który musi być spełniony w stanie przed wykonaniem operacji,
- warunek końcowy (ang. *postcondition*), który wyraża relację między stanami przed i po wykonaniu operacji.

Proces projektowania polega na dekomponowaniu abstrakcyjnych typów danych na typy prostsze, aż do poziomu elementarnych typów danych i operacji dostępnych w języku programowania. Proces przebiega stopniowo, krok po kroku, z wykorzystaniem ograniczonego zestawu struktur danych: zbiorów, list i odwzorowań oraz ograniczonego zbioru konstrukcji sterujących: sekwencji, alternatywy i iteracji. Otrzymane w każdym kroku typy danych są dokumentowane za pomocą specyfikacji identycznych jak specyfikacja początkowego problemu.

Dowód poprawności, konstruowany w każdym kroku pokazuje, że typy danych przed i po dekompozycji są homomorficzne względem warunków początkowych i końcowych. Sekwencja dowodów poprawności wszystkich kroków tworzy dowód poprawności programu względem formalnej specyfikacji wymagań.

Dla przykładu rozważmy proste zadanie rejestracji pH w reaktorze chemicznym (rys. 9). Formalna specyfikacja procedury *Aktualizuj parametry rejestracji* składa się z dwóch elementów:
- struktury danych: *PARAMETRY_REJESTRACJI*, zdefiniowanej jako odwzorowanie ze zbioru *NAZWY* w zbiór *WARTOŚCI*, oraz
- operacji: *aktualizuj (n, v)*, która wprowadza nową wartość *v* parametru *n* do definicji odwzorowania *PARAMETRY_REJESTRACJI*.

Bieżący stan procedury jest określony przez wartość odwzorowania *PARAMETRY_REJESTRACJI*. Oznaczmy przez *mp* stan przed operacją aktualizacji, a przez *mp'* stan po wykonaniu operacji. Formalną specyfikację procedury można teraz zapisać jako:

PARAMETRY_REJESTRACJI = NAZWY → WARTOŚCI
modify (n, v)
states: MONITOR-PARAMETERS
type: NAZWY, WARTOŚCI →
pre-aktualizuj(mp,n,v) = (n ∉ dom mp)
post-aktualizuj(mp.n,v,mp') = (mp' = mp⊎[n → v])

gdzie *dom* oznacza dziedzinę odwzorowania, a ⊎ jest operatorem modyfikacji wartości odwzorowania w jednym punkcie.

Podana specyfikacja nie określa ani sposobu reprezentacji danych (*NAZWY, WARTOŚCI, PARAMETRY_REJESTRACJI*), ani też algorytmu operacji. Np. dane mogą być zorganizowane jako wskaźnikowa lista par deskryptorów (nazwa – wartość) lub wektor wartości dostępnych przez skorowidz nazw. Jeżeli zdecydujemy się na drugi wariant, to operację aktualizacji można rozłożyć na sekwencję operacji prostszych:
pozycja(n,a) – pobierającą ze skorowidza adres *a* wartości parametru *n*,
zmiana(a,v) – zapisującą wartość *v* pod adres *a*.

Aby udowodnić poprawność dekompozycji należy udowodnić, że:
- sekwencja operacji jest wykonalna, tzn:

pre-aktualizuj ⇒ pre-pozycja
pre-aktualizuj & post-pozycja ⇒ pre-zmiana

- rezultaty wykonania operacji *aktualizuj* i sekwencji *(pozycja, zmiana)* są równoważne, tzn.:

pre-aktualizuj & post-pozycja & post-zmiana ⇒ post-aktualizuj

Dekompozycja i dowód jej poprawności są kontynuowane w kolejnych krokach. W chwili osiągnięcia poziomu instrukcji języka programowania, konstrukcja programu i dowodu poprawności zostają zakończone.

Pomimo swej koncepcyjnej atrakcyjności, metody formalne mają kilka słabych stron. Po pierwsze, zapisana formalnie specyfikacja wymagań jest niezrozumiała dla użytkownika, co utrudnia wiarygodną ocenę. Pewną metodą złagodzenia problemu jest wykorzystanie techniki szybkiego makietowania w języku bardzo wysokiego poziomu, np. w PROLOG-u. Po drugie, metody te mogą być efektywnie stosowane tylko w tych dziedzinach, w których istnieje naturalny model matematyczny, np. konstrukcji kompilatorów i systemów baz danych. W zastosowaniach, dla których brak takiego modelu – np. przy projektowaniu systemów wbudowanych, metody te nie są efektywne, gdyż wymagają olbrzymiego wysiłku niezbędnego dla opracowania odpowiedniej teorii matematycznej.

* * *

Praktyka inżynierska pokazuje, że najpopularniejsze i najbardziej efektywne podejście do projektowania oprogramowania opiera się na modelu tradycyjnym. Zaletami tego podejścia są: łatwość zarządzania i oceny stanu projektu programowego, dostępność sprawdzonych metod projektowych i wydajnych systemów wspomagających. Mimo tych zalet, wiele projektów programowych nie mieści się w założonych terminach i budżecie.

Szybkie makietowanie jest niezwykle użytecznym narzędziem wydatnie wspomagającym efektywność analizy i oceny założeń. Technika ta jest jednak trudna do zastosowania w bardzo dużych projektach. Nasuwa to przypuszczenie, że narzędzia wspomagające makietowanie nie są jeszcze dostatecznie efektywne.

dokończenie na s. 25

MICHAŁ GOLIŃSKI
Szkoła Główna Handlowa
Warszawa

Rynki elektroniczne – podstawowe pojęcia obszary zastosowań i aspekty ekonomiczne

Niniejszy artykuł przedstawia wybrane aspekty zastosowań informatyki i telekomunikacji w wymianie dóbr i usług. Procesy te, coraz bardziej widoczne w państwach wysoko rozwiniętych, przyczyniły się do powstania nowego terminu: rynek elektroniczny. Treścią artykułu będzie zaprezentowanie podstawowych definicji, trendów rozwojowych i ekonomicznych implikacji rozwoju rynków elektronicznych. O wyborze tematu zadecydowała jego aktualność, która jak dotąd nie znalazła odzwierciedlenia w naszym piśmiennictwie. Artykuł stanowi rekapitulację studiów literaturowych i własnych dociekań autora, poczynionych podczas stypendium w Hochschule St. Gallen w Szwajcarii.

Rozwój technologiczny ostatniego dwudziestolecia doprowadził do zbliżenia, a nawet połączenia dwóch technologii, które dotychczas istniały częściowo niezależnie od siebie – telekomunikacji i informatyki. Konwergencja obu tych dziedzin, której wynikiem jest powstanie nowego, coraz częściej używanego terminu – telematyka[1], doprowadziła do powstania interaktywnych systemów komputerowych przesyłających dane za pośrednictwem sieci telekomunikacyjnych. Wzrost mocy obliczeniowej sprzętu komputerowego wraz ze wzrostem szybkości przesyłania danych w coraz powszechniej stosowanych sieciach ISDN czy BISDN[2], umożliwiają nowe „telematyczne" zastosowania (np. przesyłanie nieruchomych, a nawet ruchomych obrazów, telekonferencje czy systemy multimedialne).

Równolegle do wspomnianego postępu technologicznego dokonała się nie mniejsza zmiana w obszarze ekonomicznym. Informacja stała się jednym z czynników produkcji, być może ważniejszym niż praca, kapitał i ziemia. Z drugiej strony obserwujemy postępującą globalizację i dynamizację rynków, która jeszcze bardziej zwiększa znaczenie „surowca informacyjnego".

Do coraz liczniejszych nowych systemów komunikacji komputerowej, takich jak: BBS-y[3], banki danych typu on-line, systemy telekonferencyjne i videotekstowe (tzw. New Telematic Services[4]) dołączyły wspomagane komputerowo systemy handlowe, realizujące transakcje kupna-sprzedaży między prywatnymi i/lub instytucjonalnymi uczestnikami rynku. Tego typu systemy zaczęto nazywać **elektronicznymi rynkami.**

Pojęcie elektronicznego rynku nie jest, jak dotychczas, jednoznacznie zdefiniowane w literaturze przedmiotu. Termin ten pojawił się w połowie lat osiemdziesiątych, w pracach grupy badawczej z MIT[5]. Różni autorzy używają go, jeśli nawet nie identycznie, to jednak dla opisu podobnej grupy zjawisk.

Dla potrzeb tego artykułu możemy przyjąć, że: *Rynek można wtedy określać jako elektroniczny, gdy zastosowanie telematyki ma ważny lub nawet decydujący o jego istnieniu wpływ na strukturę rynku i zachodzące w nim procesy* [4]. Po zinformatyzowaniu procesów inżynierskich i produkcyjnych oraz sfery zarządzania, technologie telematyczne coraz silniej wkraczają w kolejny obszar działalności człowieka – w proces wymiany dóbr i usług.

Czynnikiem, który spowodował powstanie tego typu struktur rynkowych jest oczywiście pieniądz. I tak, Hewlett-Packard, stosując EDI[6], oszczędza rocznie ponad 200 mln marek niemieckich. Producent lotniczy Douglas Airsft Company osiągnął obniżkę kosztu realizacji zamówienia standardowej części zapasowej z dziesięciu do dwóch marek. Według szacunków niemieckich, średniej wielkości firma spedycyjna, której 80% klientów jest połączonych z nią elektronicznie, może liczyć na roczne oszczędności rzędu 330 tys. marek [1].

Na podstawie badań ANSI[7] stwierdzono, że 70% wszystkich drukowanych przez komputery list i formularzy jest powtórnie wprowadzanych do innego komputera w celu dalszego przetwarzania, przy czym ponad 50% bardziej złożonych dokumentów zawiera błędy. Szacuje się, iż w państwach EWG traci się ok. 2% produktu narodowego brutto w wyniku nieadekwatnych metod składania zamówień i wielokrotnego wprowadzania tych samych informacji [7].

Podstawą każdego rynku elektronicznego jest system elektronicznej wymiany danych (EDI – *Electronic Data Interchange*).

---

[1] Termin *telematyka* (telekomunikacja + informatyka) pochodzi wprawdzie z Francji, ale jest coraz częściej stosowany również na obszarze niemieckojęzycznym (*Telematik*) oraz ma swój odpowiednik angielski (*telematics*). Jego poręczność i jasność znaczeniowa przemawiają zdaniem autora, za wprowadzeniem go do terminologii polskiej dla określenia całości zjawisk związanych z integracją przetwarzania danych i telekomunikacji. W USA często jest używany jego odpowiednik „compucations" (computing + telecommunications).

[2] ISDN – *Integrated Services Digital Network*
BISDN – *Broad-band Integrated Services Digital Network*

[3] BBS – *Bulletin Board System*

[4] The development of New Telematic Services. OECD STI Reviev, Vol 7, 1990 (w [5])

[5] Mallone T. W., Yates J., Benjamin R. I.: Electronic Markets and Electronic Hierarchies. MIT, 1986 (w [5])

[6] EDI – *Electronic Data Interchange* (elektroniczna wymiana danych) – powszechnie zaakceptowany standard elektronicznej wymiany dokumentów handlowych.

[7] ANSI – *American National Standards Institute.*

W początkowej fazie rozwoju tego typu systemów, uczestniczący w nich dostawcy i odbiorcy informacji ustanawiali własne normy dokumentów i protokoły transmisji. Jednak zachodzące w tej sferze procesy integracyjne (przystępowanie do tego rynku uczestników spoza branży, łączenie się dotychczas odrębnych segmentów rynku w jeden) zrodziły potrzebę działań standaryzacyjnych. Szybko przekonano się o konieczności stworzenia jednolitego, międzynarodowego standardu EDI. W drugiej połowie lat osiemdziesiątych narodził się, wspierany przez ONZ, projekt EDIFACT – *Electronic Data Interchange for Administration, Commerce and Transport,* stanowiący zbiór norm i standardów, którymi posługują się wszyscy tworzący współczesne systemy rynków elektronicznych.

Elektronizacja rynków

Rynek możemy zdefiniować jako miejsce, gdzie spotykają się osoby i(lub) instytucje (uczestnicy rynku) w celu wymiany określonych dóbr i(lub) usług. Pojęcie rynku nie musi być związane z konkretnym miejsce i czasem.

Podstawowe struktury

Ritz i Schnückel [6] wyróżniają dwa podstawowe mechanizmy koordynujące przepływ strumieni dóbr i usług w gospodarce. Są to rynki i hierarchie.

Rynek koordynuje przepływ dóbr i usług za pomocą oddziaływujących na nim mechanizmów popytu i podaży, kształtujących charakter transakcji zachodzących między uczestnikami rynku. Wspomniane siły rynkowe determinują charakter, cenę, ilość oraz termin i sposób dostawy danego dobra czy usługi. Kupujący na podstawie najdogodniejszej dla niego kombinacji wymienionych czynników dokonuje wyboru między poszczególnymi ofertami.

Hierarchia koordynuje przepływ dóbr i usług za pomocą instancji kontrolującej i nadzorującej. Instancja ta decyduje w sposób arbitralny o charakterze oferty. Kupujący nie może swobodnie dokonać wyboru dostawcy. Zostaje on zmuszony przez instancję nadzorującą do współpracy z konkretnym partnerem.

Podana typologia ma oczywiście charakter idealny, a struktury występujące w świecie realnym mają najczęściej charakter mieszany. Będzie ona jednak przydatna do objaśnienia niektórych poniżej opisanych procesów.

Do czynników determinujących powstanie jednej z tych struktur zalicza się, przede wszystkim, koszty produkcji i koszty koordynacji. Koszty produkcji, to wszystkie koszty pierwotne, niezbędne do wytworzenia i dystrybucji dóbr czy usług. Koszty koordynacji, to koszty przetwarzania informacji wszystkich transakcji, niezbędnych do koordynacji czynności uczestniczących w procesie produkcji.

Z punktu widzenia kupującego, koszty produkcji i koszty koordynacji zmieniają się w zależności od charakteru struktury regulującej przepływ dóbr i usług. Charakter tej zależności ilustruje tabela.

Kupujący w strukturze rynkowej jest w stanie dokonać wyboru najkorzystniejszego dla siebie rozwiązania, wyszukując i porównując wiele ofert. Pozwala to zminimalizować koszty produkcji. Z drugiej jednak strony, czasochłonność oraz wysokie koszty procesów poszukiwania i wyboru powodują, że koszty koordynacji w strukturze rynkowej są wysokie.

Hierarchia narzuca kupującemu warunki zakupu. Koszty produkcji są w tej sytuacji najczęściej wyższe (brak możliwości wyboru oferty optymalnej). Jednak koszty koordynacji będą niższe, gdyż zostaje wyeliminowana konieczność poszukiwania i dokonywania wyboru.

Zależność kosztów produkcji i koordynacji od charakteru przepływu dóbr i usług

Struktura	Koszty produkcji	Koszty koordynacji
Rynek	niskie	wysokie
Hierarchia	wysokie	niskie

Źródło: [6]

Etapy rozwoju rynków elektronicznych

Podane poniżej etapy rozwojowe rynków elektronicznych mają oczywiście charakter umowny i orientacyjny. Opierają się na wprowadzonej poprzednio typologii struktur. W praktyce istnieje możliwość, że podlegający procesowi elektronizacji rynek pominie w swym rodzaju któryś z etapów lub będzie rozwijał się według innego scenariusza.

Elektroniczna hierarchia, to systemy informatyczno-telekomunikacyjne, łączące pojedynczego oferującego (najczęściej twórcę i jednocześnie prowadzącego system) z wieloma kupującymi. Pierwsze tego typu systemy pojawiły się w latach siedemdziesiątych, a ich najlepszym przykładem są systemy rezerwacji i zamówień. Z jednej strony ułatwiły one kupującemu dokonywanie zamówień i prowadzenie gospodarki materiałowej, z drugiej jednak czyniły go zależnym od jednego dostawcy. Cele, jakie przyświecały twórcom takich systemów, to silniejsze związanie ze sobą klientów oraz wyeliminowanie konkurentów lub przynajmniej stworzenie barier utrudniających konkurencji wejście na rynek.

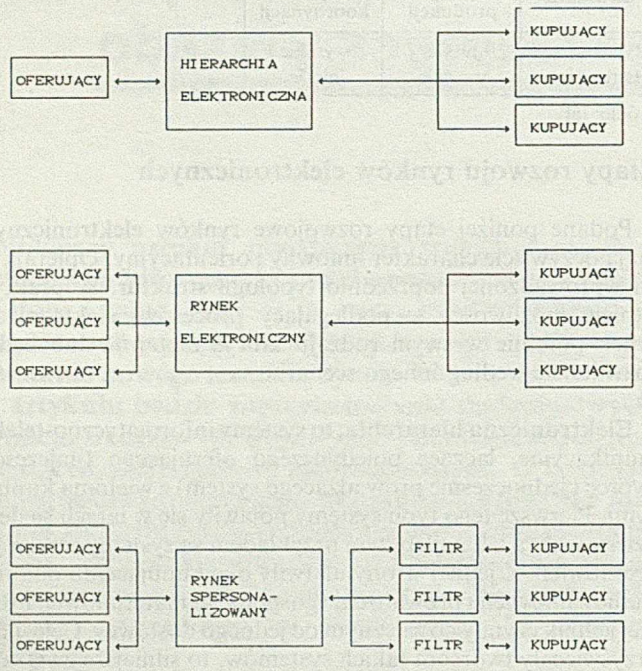
Rynkiem elektronicznym jest system informatyczno-telekomunikacyjny (nawet gdy jest on prowadzony przez pojedynczą firmę), który łączy wielu oferujących z wieloma kupującymi. Oferującym pozwala dotrzeć do szerszego kręgu potencjalnych klientów, natomiast kupującym – dokonać wyboru spośród wielu ofert. Większość rynków elektronicznych wykształca się z elektronicznych hierarchii na drodze ewolucyjnej pod wpływem sił rynkowych. Proces ten rozpoczyna się najczęściej wtedy, gdy w określonym obszarze rynku konkurenci zaczynają równolegle tworzyć własne elektroniczne kanały dystrybucji. Tendencja przechodzenia do struktur rynkowych jest wspomagana także przez czynniki legislacyjne oraz organy administracji państwowej czuwające nad ochroną wolnej konkurencji[8].

Rozwinięty rynek elektroniczny oferuje kupującemu znaczną liczbę i różnorodność ofert. Częstym skutkiem tego bogactwa ofert jest swoiste „zatrucie informacyjne" (ang. *information overload*) kupującego, zmuszonego do przejrzenia i oceny wszystkich możliwości. Problem ten staje się szczególnie istotny na tych rynkach, gdzie są oferowane dobra czy usługi mogące być rozmaicie oceniane w zależności od struktury preferencji kupującego. Wyjściem z tej sytuacji byłoby stworzenie spersonalizowanych „pomocy decyzyjnych", redukujących na początek liczbę możliwych wariantów. W następnym etapie możliwym wydaje się wmontowanie w system „inteligentnych filtrów uaktywnianych w celu redukcji zatrucia informacyjnego. W dalszej przyszłości można wyobrazić sobie udostępnianie oferują-

[8] Dobrym przykładem jest tu decyzja *Civil Areonautics Board* z 1983 r., uznająca za pogwałcenie zasad wolnej konkurencji postępowanie linii lotniczych American Airlines, które udostępniały biurom podróży swój system komputerowej rezerwacji miejsc APOLLO tylko dla zamawiania własnych lotów.

cym, oczywiście w anonimowej formie, swoistych profili preferencji klientów. Byłoby to niezwykle efektywne narzędzie badania rynku, pozwalające producentom lepiej dopasowywać swe oferty do potrzeb odbiorców. Tego typu struktura to **rynek spersonalizowany.**

Rys. 1 przedstawia, w schematyczny sposób, trzy wyżej omówione, etapy rozwoju elektronicznych rynków.



Rys. 1

Efekty ekonomiczne elektronizacji rynków

Zastosowanie technik telematycznych w procesach wymiany dóbr i usług wywołuje konkretne, wymienione poniżej, efekty ekonomiczne.

Efekt komunikacyjny – wyrażający się zarówno obniżką kosztów przesyłania, przypadających na jednostkę informacji, jak i skróceniem czasu transmisji.

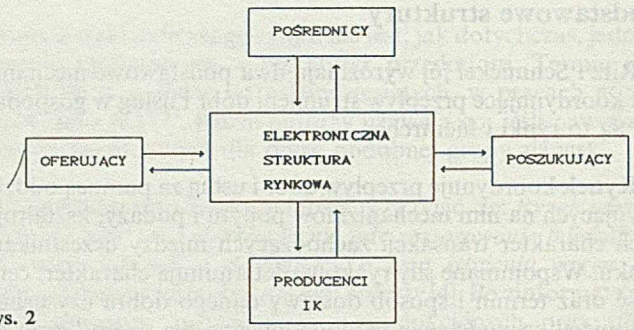
Efekt brokera – w wyniku którego rynek elektroniczny spełnia funkcje pośrednika łączącego potencjalnych partnerów rynkowych umożliwiając im dokonywanie porównań między większą liczbą ofert, poprawę jakości dokonywanego wyboru oraz zmniejszenie kosztów koordynacji.

Efekt integracyjny – (obecnie najbardziej widoczny w branży motoryzacyjnej) – pojawiający się wtedy, gdy partnerzy rynkowi wykorzystują integrację komunikacyjną (IK) nie tylko jako sposób na uproszczenie procesów komunikacyjnych, lecz postępują krok dalej. Dążą do przyspieszenia i integracji procesów zbierania i przetwarzania informacji, co pozwala:

- wyeliminować konieczność wielokrotnego wprowadzania tych samych danych przez obu partnerów, powodując obniżkę kosztów w wyniku oszczędności czasu oraz redukcji liczby błędów;
- stworzyć możliwość realizacji procesu produkcji bezmagazynowej (*just-in-time*), co przy połączeniu systemów gospodarki magazynowej obu partnerów pozwala zmniejszyć zapasy produkcyjne oraz znacznie zredukować koszty;
- prowadzić wspólne prace badawcze i konstrukcyjne dzięki połączeniu systemów CAD/CAM producenta i klienta; umożliwia to testowanie nowych produktów przez odbiorcę już w fazie projektowania, korzystanie z tych samych danych badawczo-rozwojowych, a co za tym idzie, przygotowanie produktu najbardziej odpowiadającego potrzebom odbiorcy.

Potencjalni uczestnicy rynku elektronicznego

Oprócz pierwotnych partnerów rynkowych, których istnienie jest niezbędne by móc mówić o strukturach rynkowych (tzn. oferującego i poszukującego), uczestnikami rynku elektronicznego będą najczęściej jeszcze dwie kategorie uczestników. Jedną z nich są wszelkiego rodzaju organizacje pośredniczące – brokerzy, agenci i pośrednicy uczestniczący w procesie transferu dóbr i usług, drugą – producenci sprzętu IK, udostępniający sprzęt i oprogramowanie, niezbędne do elektronicznego zawierania transakcji. Rys. 2 przedstawia wymienione grupy uczestników.



Rys. 2

Głównym motywem powodującym, że oferujący dobra lub usługi przystępuje do elektronicznego rynku, jest chęć ściślejszego związania z sobą odbiorcy. Pozwoli to na stworzenie zabezpieczonych i długotrwałych kanałów dystrybucji dla własnych produktów. Cel ten można osiągnąć wtedy, gdy oferujący udostępni swoim klientom elektroniczny system realizacji zamówień, umożliwiając im w ten sposób racjonalizację procesów zaopatrzenia, a zwłaszcza uproszczenie i zredukowanie zakresu gospodarki materiałowej. Tego typu systemy rezerwacji lub zamówień są już dziś bardzo rozpowszechnione, jednak z racji dominującej pozycji oferującego, mają one najczęściej charakter hierarchii.

Głównym motywem skłaniającym odbiorców do przystąpienia do systemu rynku elektronicznego jest chęć wyboru spośród jak największej liczby ofert przy jak najmniejszym nakładzie środków na wyszukiwanie i ocenę oferty optymalnej.

Z inną sytuacją mamy do czynienia wtedy, gdy inicjatywa stworzenia elektronicznej struktury rynkowej leży w interesie odbiorcy. Systemy tego typu pozwalają zracjonalizować wybór konkretnego dostawcy, zwiększyć efektywność procesów zaopatrzenia oraz zminimalizować zapasy magazynowe. Wiele procesów gospodarczych, a więc także związane z nimi koszty, zostaje w ten sposób przeniesionych na dostawców.

W niektórych wypadkach inicjatywa stworzenia elektronicznej struktury rynkowej pochodzić może od organizacji pośredniczących w dystrybucji dóbr i usług, takich jak wielcy hurtownicy, brokerzy informacyjni czy agenci. Wystąpi to zwłaszcza wtedy, gdy perspektywa bezpośredniego połączenia dostawców i odbiorców stwarza niebezpieczeństwo wyeliminowania ich z rynku. W tej sytuacji dalszą racją bytu dla pośredników może

być stworzenie dogodnego dla wszystkich uczestników systemu rynku elektronicznego.

Istotną rolę w systemach rynków elektronicznych odgrywają producenci IK. Dla technicznej realizacji elektronicznych struktur rynkowych jest konieczny dostęp do wyspecjalizowanego sprzętu i oprogramowania (sieci, serwery, oprogramowania komunikacyjne itp.). Realizacja systemów tego typu jest zlecana wyspecjalizowanym firmom z branży informatycznej i telekomunikacyjnej. W tej sytuacji, naturalną tendencją tych firm jest przechodzenie od pasywnej roli realizującego system do roli tworzącego i utrzymującego na zasadach komercyjnych system rynku elektronicznego.

Możliwości przystosowania dóbr i usług do elektronicznego rynku

Nie wszystkie dobra i usługi kwalifikują się w takim samym stopniu do tego, aby być przedmiotem obrotu na rynkach elektronicznych. Najwięcej problemów (przynajmniej przy dzisiejszym stanie techniki) przysparzają dobra i usługi trudno opisywalne i silnie zindywidualizowane.

Dobra i usługi mogą być przedmiotem dystrybucji w elektronicznych systemach rynkowych wtedy, gdy potencjalny kupujący jest w stanie podjąć decyzję o zakupie na podstawie odpowiednich, zawartych w systemie informacji (ilościowych i jakościowych) o danym produkcie. Najłatwiej dają się „elektronizować" transakcje, których przedmiotem są dobra standaryzowane i masowe, łatwe do precyzyjnego opisania i specyfikacji, takie jak akcje, czy ropa naftowa. Dzisiejszy stan techniki IK nie pozwala jeszcze efektywnie i tanio przesyłać złożonych danych, takich, jak rysunki konstrukcyjne z systemów typu CAD czy obrazy wideo. Jednak dalszy rozwój techniki przesyłania informacji, a zwłaszcza wykorzystanie szerokopasmowego ISDN oraz technik multimedialnych, pozwolą w przyszłości włączyć do elektronicznego obrotu nowe rodzaje dóbr i usług.

Zrealizowane elektroniczne struktury rynkowe

Poniżej zostaną scharakteryzowane przykłady już zrealizowanych elektronicznych struktur rynkowych.

System SABRE AMERICAN AIRLINES

Systemy rezerwacji lotów należą do pierwszych realizacji elektronicznych struktur rynkowych. Jako pierwsza, we wczesnych latach siedemdziesiątych, system taki zrealizowała amerykańska linia lotnicza *United Airlines*. System ten, o nazwie APOLLO, miał jednak charakter hierarchii, ponieważ umożliwiał rezerwację lotów tylko tego jednego towarzystwa. W odpowiedzi na to, konkurencyjne towarzystwo lotnicze – *American Airlines* uruchomiło podobny system o nazwie SABRE, pozwalający jednak na dokonywanie rezerwacji lotów także w innych towarzystwach. Była to niewątpliwie pierwsza realizacja systemu rynku elektronicznego. Warto zaznaczyć, że w okresie późniejszym United Airlines, pod naporem konkurencji oraz w wyniku problemów prawnych, uczynił APOLLO także systemem ogólnodostępnym.

System ASAP

Firma, *American Hospital Supply* (AHS), zajmująca się zaopatrzeniem szpitali wprowadziła na początku lat siedemdziesiątych elektroniczny system dystrybucji oferowanych towarów ASAP. AHS dostarcza wyroby prawie 8500 producentów ponad 100 000 odbiorców. Klienci AHS zamawiają potrzebne dobra za pomocą terminala, co znacznie redukuje koszty zakupu i umożliwia minimalizację zapasów magazynowych. Korzyści, jakie uzyskała firma AHS z takiej formy dystrybucji oferowanych przez siebie produktów, są następujące:

- zmniejszenie własnych zapasów magazynowych,
- poprawa jakości i efektywności obsługi klienta,
- zwiększenie obrotów w wyniku możliwości szybkiego wynegocjowania od własnych dostawców korzystniejszych warunków sprzedaży, a więc wzrost konkurencyjności cenowej wobec odbiorców,
- znaczne utrudnienie wejścia na rynek dla konkurencji,
- zacieśnienie więzi z klientem,
- łatwość dostępu do danych o klientach, a więc szybkie analizowanie trendów rynkowych i reagowanie na zmiany potrzeb klientów.

Standardy amerykańskiej branży samochodowej

W 1981 r. czterech największych producentów samochodów w USA oraz ich 300 największych dostawców stworzyło organizację o nazwie *Automotive Industry Action Group* (AIAG). Celem tej organizacji było stopniowe zastąpienie ogromnej masy papieru (rachunki, oferty, zamówienia itp.), przepływającej podczas wewnątrzbranżowej komunikacji między poszczególnymi firmami, ze standaryzowanym elektronicznym systemem wymiany informacji. Miało to znacznie zmniejszyć koszty (dotychczasowa „papierowa" technologia kosztowała rocznie ok. 2 mld dolarów) oraz przyczynić się do zwiększenia szybkości wymiany informacji między użytkownikami systemu. 25 000 dostawców przemysłu samochodowego w USA zobligowano do adaptacji w ciągu trzech lat ustalonych standardów. Firmy, które tego nie dokonały zostały z rynku wykluczone. Wprowadzenie elektronicznego systemu komunikacji przyczyniło się nie tylko do obniżenia kosztów oraz przyspieszenia procesu wymiany informacji, ale także stworzyło podstawy do wprowadzenia obecnie już powszechnej produkcji bezmagazynowej.

Podobne cele przyświecały stworzonej w 1984 r. przez dziewięciu głównych europejskich producentów branży samochodowej analogicznej organizacji o nazwie ODETTE (*Organization for Data Exchange by Teletransmission*).

System PRONTO

Amerykański *Chemical Bank*, wprowadzając w 1983 r. system PRONTO, był jednym z pionierów dyspozycji bankowej z domu (*Home-Banking*). Już w 1985 r. zarejestrowano ok. 20 000 użytkowników tego systemu, będącego częścią TARGET *Information and Transaction Network*, stworzonego również przez *Chemical Bank* wspólnie z koncernem telekomunikacyjnym AT&Ta, przeznaczonego do obsługi zarówno osób prywatnych, jak i małych przedsiębiorstw.

System PRONTO oferuje następujące usługi:

- podawanie informacji dotyczących aktualnego stanu konta bankowego oraz dokonywanie transferu pieniędzy na inne konta,
- realizacja płatności, zarówno jednokrotnych jak i okresowych,
- możliwość śledzenia zmian stanu konta,
- możliwość dokonywania takich operacji, jak otwieranie oraz blokowanie kont (podsystem *Self-Service Banking Center*),
- dostarczanie w formie elektronicznej artykułów z czasopisma „Money", periodyku poświęconego giełdzie i formom oszczędzania (podsystem *Money On-line*),
- dostarczanie informacji konsumenckich, zawierających informacje podatkowe, wyniki testów itp.,

- dostarczanie informacji o aktualnych wydarzeniach kulturalnych, programach kin i teatrów, restauracjach itp.

PRONTO można zainstalować na większości popularnych komputerów osobistych. Użytkownik nie dysponujący takim komputerem otrzymuje terminal AT&T. System jest bardzo przyjazny dla użytkownika i umożliwia łatwy wybór odpowiadającego użytkownikowi trybu pracy.

System INVESTDATA

Szwajcarska firma *Telekurs,* prowadząca wspólne centrum obliczeniowe dla rozliczeń międzybankowych banków szwajcarskich, jest także jednym z największych na świecie producentów informacji finansowej. Z bankiem danych systemu o nazwie INVESTDATA połączono ponad 80 giełd papierów wartościowych, 18 giełd surowcowych oraz znaczną liczbę banków i kreatorów rynku[9)], umożliwiając w ten sposób w każdej chwili dostęp do aktualnych informacji o wydarzeniach na rynkach światowych.

System INVESTDATA oferuje użytkownikowi m.in. aktualne dane na temat:
- kursów ok. 160 000 akcji i obligacji,
- posunięć głównych kreatorów rynku,
- cen ok. 40 000 opcji i transakcji terminowych,
- cen surowców z 18 giełd,
- podstawowych informacji dotyczących ponad 200 000 tematów finansowych,
- kursów metali szlachetnych i dewiz,
- ogólnych informacji ekonomicznych oraz wydarzeń międzynarodowych.

Systemem posługują się najczęściej użytkownicy profesjonalni – banki, domy maklerskie, duże firmy oraz agencje informacyjne i wydawcy gazet. Zwiększa się jednak liczba prywatnych, aktywnych inwestorów, którzy do komunikacji z systemem wykorzystują wolniejszy, ale i tańszy system videotekstowy.

Cały współczesny rynek finansowy, na którym dosłownie w ciągu minut są sprzedawane lub kupowane akcje, obligacje, opcje czy transakcje terminowe o wartości wielu miliardów dolarów, zlokalizowane na wielu giełdach świata, jest już rynkiem elektronicznym. Bez takich systemów informacji ekonomicznej, jak REUTERS, TELERATE, BLOOMBERG czy międzybankowy SWIFT, nie mógłby on funkcjonować w takiej formie, skali i tak sprawnie, jak obecnie.

Skutki ogólnogospodarcze elektronizacji rynków

Postępujące usieciowienie uczestników rynku pozwala już dzisiaj zaobserwować zarówno zmiany zachodzące w poszczególnych segmentach rynku, jak i zmiany zaistniałe na poziomie całej gospodarki. Ekonomia nie dopracowała się dotychczas żadnych konkretnych teorii dotyczących wpływu elektronizacji rynku na przebieg procesów gospodarczych. Poniżej podano i krótko scharakteryzowano zaledwie kilka tendencji, jakie można już obecnie zaobserwować w gospodarkach, w których występuje zjawisko elektronizacji rynków. Tendencje te dotyczyć będą alokacji zasobów oraz nowego charakteru konkurencji.

Jak już wspomniano, stworzenie elektronicznych powiązań między producentami, a odbiorcami powoduje znaczne redukcje kosztów. Polepsza to sytuację przedsiębiorstwa wobec konkurentów spoza rynku elektronicznego, gdyż może ono szybciej dostosować produkcję do zmieniających się potrzeb odbiorców oraz odnosić korzyści z rosnącej skali produkcji. Pozostałe firmy, chcąc utrzymać konkurencyjność swych produktów, zostaną zmuszone do wprowadzenia nowej techniki. Tym, którzy tego nie zrobią, grozi znaczne osłabienie ich pozycji rynkowej.

Techniki IK stały się ogólnodostępnym dobrem, dostarczanym przez wyspecjalizowane firmy na ogromny, obecnie, rynek teleinformatyczny[10)]. To przesunięcie nakładów na badania i rozwój z firm produkcyjnych do wyspecjalizowanych firm z branży IK powoduje powszechną dostępność takich systemów, umożliwiając także małym i średnim przedsiębiorstwom stworzenie elektronicznych połączeń ze swymi partnerami rynkowymi. Może to prowadzić do zwiększenia stopnia konkurencji, ponieważ małe firmy uzyskają możliwość wejścia na rynek elektroniczny i zajęcia na nim nisz rynkowych, pozostawionych przez przedsiębiorstwa wielkie.

Badając funkcjonujące obecnie hierarchie rynkowe ukierunkowane na odbiorców, możemy zauważyć, że zapasy magazynowe przedsiębiorstw produkcyjnych ulegają zmniejszeniu, natomiast rosną zapasy magazynowe dostawców. Należy przypuszczać, iż dostawcy, zmuszeni do uczestnictwa w bezmagazynowej produkcji odbiorcy, będą dążyli do zepchnięcia tej uciążliwości na poprzedzające ich ogniwo łańcucha produkcyjnego. Wynikiem tej tendencji będzie ogólne przyspieszenie procesów produkcyjnych, zmniejszenie się zapasów materiałowych oraz zmniejszenie zużycia surowców (wyeliminowany zostaje problem „zestarzenia" się zapasów) w skali całej gospodarki.

Tendencje rozwojowe

Siła mechanizmów rynkowych, a w wielu krajach także rozwiązania prawne, będą powodować tendencje do przechodzenia od elektronicznych hierarchii do elektronicznych rynków.

Istnienie efektywnego rynku elektronicznego skłoni wielu nabywców do rozluźnienia dotychczasowych, bliskich związków z jednym dostawcą na korzyść możliwości szybkiego i elastycznego reagowania na atrakcyjne oferty innych dostawców.

Zrealizowane dotychczas systemy są silnie związane z tradycyjnym przebiegiem procesów rynkowych. Wspomagają one najczęściej tylko małe, wyraźnie rozgraniczone segmenty rynku. Liczba uczestników takiego rynku jest najczęściej stosunkowo mała (często są to członkowie określonej organizacji lub grupy interesów, np. producenci z branży samochodowej).

Zastosowanie nowych technologii pozwoli na zmianę przebiegu procesów rynkowych oraz form organizacyjnych rynku. Już dzisiaj, opierając się na technologii IK, można zorganizować rynek w oderwaniu od konkretnego miejsca. Inną tendencją będzie prawdopodobnie integracja segmentów rynku obsługiwanych dotąd przez różne systemy. Powstałby w ten sposób swoisty „elektroniczny jarmark", na którym nabywca mógłby kupować potrzebne mu dobra i usługi bez konieczności używania wielu odrębnych systemów. Zredukowałoby to, konieczne jeszcze dziś, znaczne nakłady, jakich wymaga nauczenie się obsługi wielu, często niekompatybilnych, systemów. To, z kolei,

[9)] *Kreator (twórca) rynku – Market Maker* – jest to wg [8] uznana instytucja finansowa lub osoba fizyczna konsekwentnie i ciągle oferująca kupno i sprzedaż wybranych instrumentów finansowych na rynku wtórnym

[10)] Diebold szacował w 1989 r. dla RFN wielkość tego rynku na 155 mld marek [3].

zwiększyłoby atrakcyjność rynku elektronicznego dla zwykłego konsumenta, pozyskując dla tego rynku nowych uczestników.

Implikacje dla potencjalnych uczestników rynku

Wyliczone skutki procesu elektronizacji rynków nie wyczerpują całości tego zagadnienia. Podane niżej hasła sygnalizują tylko niektóre z głównych problemów, przed jakimi staną w przyszłości uczestnicy gry rynkowej:

- Istnienie elektronicznych rynków jest już obecnie faktem, a ich znaczenie w przyszłości będzie rosło. W sposób pośredni lub bezpośredni będą one wpływały na działalność każdego przedsiębiorstwa. Fakt ich istnienia musi więc być wkalkulowany w strategię rozwoju firmy.

- Każdy producent musi dokonać gruntownej analizy i odpowiedzieć na pytanie, czy oferowane produkty lub usługi mogą być dystrybuowane na rynku elektronicznym. Jeśli tak, to w jakiej formie (stworzenie i utrzymywanie własnego systemu, czy integracja na jednym z istniejących rynków)?

- Producent powinien opracować strategię działania na wypadek, gdyby stworzona przez jednego z konkurentów hierarchia elektroniczna odcięła firmę od rynku lub znacznie ograniczyła do niego dostęp.

- Organizacje pośredniczące powinny zwracać uwagę, aby stworzenie bezpośrednich elektronicznych związków między producentami a końcowymi odbiorcami nie odebrało im racji bytu. Ich szansą może być współudział w tworzeniu rynku elektronicznego i dalsze odgrywanie roli pośredniczącej,

- Przed firmami produkującymi oprogramowanie otwiera się duży rynek inteligentnych pomocy decyzyjnych, które będą niezbędne na etapie rynku spersonalizowanego,

- Działy informatyki w przedsiębiorstwach produkcyjnych powinny zająć się planowaniem niezbędnej dla opisanych systemów infrastruktury sieciowej oraz przygotować odpowiedni potencjał rozwojowy.

W skali światowej obserwuje się obecnie zjawisko globalizacji rynków oraz ich swoiste „odmiejscowienie" i „odczasowienie". Rynek elektroniczny nie jest związany z żadnym konkretnym miejscem ani porą dnia – jest wszędzie, gdzie sięgają łącza telekomunikacyjne i to przez całą dobę. Systemy tego typu na świecie już działają i jakkolwiek w Polsce będziemy musieli na tego typu zjawiska jeszcze poczekać[11] ze względu na zły stan telekomunikacji, kryzys gospodarczy, zacofanie technologiczne, a może przede wszystkim organizacyjne. Dlatego też przedsiębiorcy polscy, chcąc współpracować z firmami zachodnimi, muszą zdawać sobie sprawę z tego, że pewnego dnia natkną się na zupełnie nową i bardzo trudną barierę, do pokonania której powinni odpowiednio się przygotować.

Elektronicznie złączony świat jutra powinniśmy rozumieć nie tylko jako proste ulepszenie stanu istniejącego. Musimy przygotować się na zupełnie nowe struktury rynkowe oraz zmienione przepływy dóbr i usług w obrębie całej gospodarki.

[11] Jako przykład rynku elektronicznego funkcjonującego w Polsce można podać tę część zagranicznych transakcji międzybankowych, która jest realizowana z wykorzystaniem systemu SWIFT (*Society for Worldwide Interbank Financial Telecommunication*), do którego należy już kilkadziesiąt polskich banków.

LITERATURA

[1] EDI wird zur Existenzfrage. W: Diebold Management Report No. 7/90. Diebold Deutschland GmbH, Eschborn, 1990

[2] Himberger A., Reck M., Schmid M.: Telematisierung von Märkten. Institut für Wirtschaftsinformatik HSG, St. Gallen, 1990

[3] Markt-Forum '89 – Der Markt für Informations – und Kommunikationstechnik in der Bundesrepublik Deutschland. Diebold Deutschland GmbH, Eschborn, 1989

[4] Positionierung und Darstellung des Problembereichs „Elektronische Märkte". Kompetenzzentrum Elektronische Märkte, Institut für Wirtschaftsinformatik HSG, St. Gallen, 1990

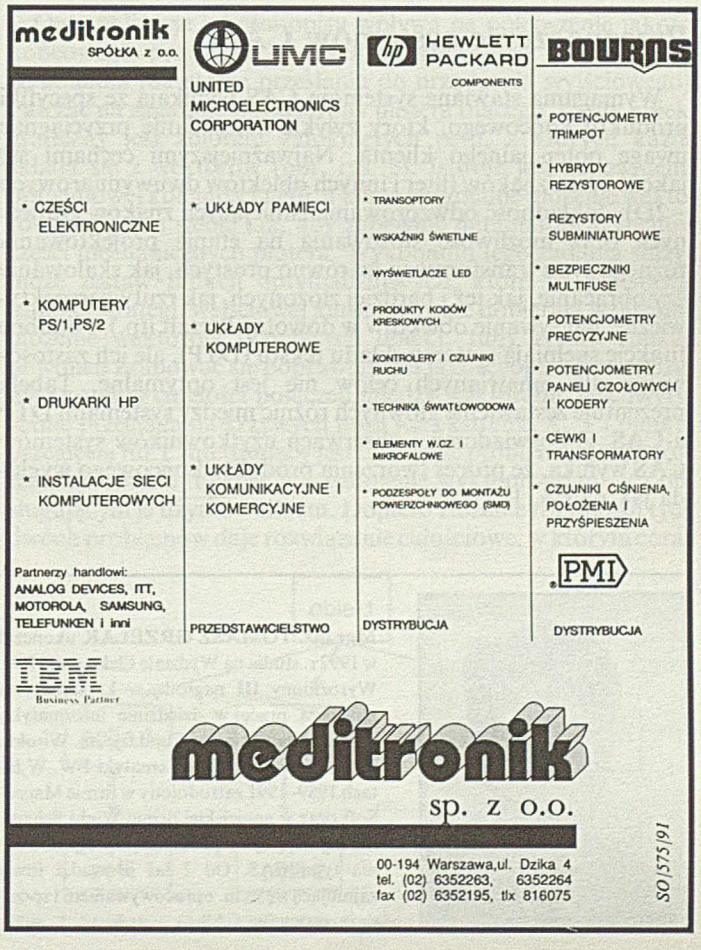
[5] Ritz D.: Entstehungsmuster und Entwicklungrichtungen Elektronischer Marktsysteme. Institut für Wirtschaftsinformatik HSG, St. Gallen, 1991

[6] Ritz D., Schnückel P.: Ökonomische Aspekte elektronischer Märkte – eine Bestandesaufnahme aus betriebs – und volkswirtschaftlicher Sicht. Institut für Wirtschaftsinformatik HSG, St. Gallen, 1990

[7] Schmid M.: Elektronische Märkte auf der Basis von Electronic Data Interchange. Institut für Wirtschaftsinformatik HSG, St. Gallen, 1990.

W związku z wprowadzeniem powszechnego podatku dochodowego i koniecznością informowania właściwych Urzędów Skarbowych o wszystkich wypłatach zwracamy się do Autorów wysyłających teksty do opublikowania o podawanie następujących danych:

- **nazwisko, imiona (pierwsze i drugie),**
- **imiona: ojca i matki,**
- **miejsce i data urodzenia,**
- **numer identyfikacyjny PESEL (wpisany przez Biuro Meldunkowe do dowodu osobistego – nie mylić z numerem dowodu osobistego!),**
- **dokładny adres (miejsce zameldowania),**
- **adres Urzędu Skarbowego właściwego dla miejsca zamieszkania Autora (bardzo ważne – bez tej informacji nie możemy przekazać honorarium do wypłaty!).**



TOMASZ GRZELAK
Zakład Grafiki Komputerowej
Instytut Informatyki
Politechnika Warszawska

Systemy wspomagania projektowania w przemyśle reklamowym

Jedną z gałęzi zastosowań grafiki komputerowej są systemy wspomagania projektowania kompozycji graficznych dla tzw. sign industry, branży usług zajmującej się produkcją wszelkiego rodzaju tablic informacyjnych, reklamowych, wystroju wystaw itp.

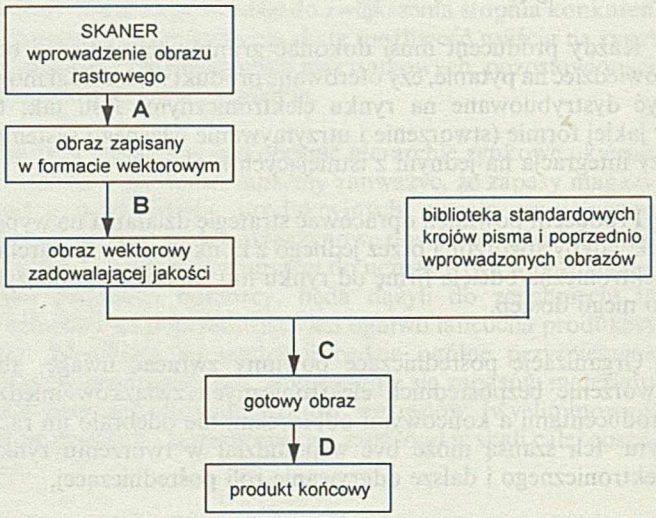
Typowy system typu CAS (*Computer Aided Signmaking*) składa się z programów wspomagających projektowanie oraz programów sterujących urządzeniem wykonującym końcowy produkt (ploter tnący folię samoprzylepną lub płyty plastikowe, urządzenie malujące powierzchnie).

Niniejszy artykuł omawia cechy i przeznaczenie systemów CAS, ze szczególnym uwzględnieniem funkcji wspomagania projektowania kompozycji graficznych. Omówione zostaną przykłady rozwiązań istniejących oraz koncepcja nowego systemu tego typu.

Przeznaczenie systemów CAS

Wymagania stawiane systemom CAS wynikają ze specyfiki produktu końcowego, który zwykle ma zadanie przyciągnąć uwagę potencjalnego klienta. Najważniejszymi cechami są: jakość opisu znaków (liter i innych obiektów dwuwymiarowych – 2D), dokładność odwzorowania istniejących znaków graficznych oraz możliwość stosowania na etapie projektowania różnego typu transformacji (zarówno prostych, jak skalowanie czy obracanie, jak też i bardziej złożonych, jak rzuty perspektywiczne, wpisywanie obiektów w dowolny kształt itp.). Podobne funkcje spełniają systemy składu tekstu (DTP), ale ich zastosowanie dla omawianych celów nie jest optymalne. Tabela prezentuje zestawienie głównych różnic między systemami DTP a CAS. Z doświadczeń i obserwacji użytkowników systemów CAS wynika, że proces tworzenia produktu końcowego wygląda jak na rys. 1.

Mgr inż. TOMASZ GRZELAK ukończył w 1992 r. studia na Wydziale Elektroniki PW. Wyróżniony III nagrodą w konkursie na najlepszą pracę w dziedzinie informatyki, organizowanym przez Fundację im. Witolda Lipskiego i Instytut Matematyki PW. W latach 1989–1991 zatrudniony w firmie Macro-Soft oraz w angielskiej firmie Wight Scientific, zajmującej się produkcją oprogramowania typu CAS. Od 2 lat prowadzi firmę zajmującą się m.in. opracowywaniem i sprzedażą systemów CAS.



Rys. 1. Proces tworzenia produktu końcowego w przemyśle reklamowym; poszczególne etapy wykonania produktu końcowego oznaczają: A – automatyczny proces wektoryzacji, B – proces wprowadzania poprawek do obrazu wektorowego, C – projektowanie końcowej kompozycji graficznej, D – przesłanie do urządzenia wyjściowego

Zestawienie cech systemów składu tekstu (DTP) i systemów CAS

	DTP	CAS
Produkt końcowy	Zwykle całe strony tekstu, docelowe litery i małe znaki graficzne	Krótkie teksty, często poddane przekształceniom graficznym, docelowe litery i duże znaki graficzne
Typ urządzenia wyjściowego	Urządzenie rastrowe	Urządzenie wektorowe
Sposób opisu obiektów graficznych	Opis rastrowy lub wektorowy, znaczny margines dopuszczalnej niedokładności opisu, gdyż produkt końcowy jest zwykle niewielkich rozmiarów	Opis wektorowy, wczytane obrazy rastrowe muszą być zwektoryzowane, bardzo istotna jest dokładność opisu ze względu na dopuszczalne duże rozmiary produktu końcowego

Przegląd cech wybranych systemów CAS

Wszystkie systemy typu CAS są dostarczane wraz ze standardowym zbiorem krojów pisma. Z punktu widzenia użytkownika istotne jest, aby istniała możliwość wprowadzania do tych krojów poprawek, jak również tworzenia krojów nowych. Spełnienie tego postulatu wymaga wprowadzenia jednolitego formatu dla obrazów uzyskanych w procesie wektoryzacji i plików z krojami pisma. Daje to ponadto wygodę w projekto-

waniu końcowej kompozycji graficznej (rys. 1C), gdyż jej wszystkie elementy składowe są wówczas traktowane jednakowo.

Poniżej zostaną omówione te cechy systemów CAS, które w decydującym stopniu wpływają na ich użyteczność.

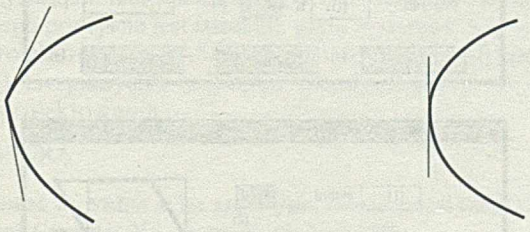
- Możliwość wczytywania obrazów rastrowych i ich wektoryzacji (rys. 1A). Istotna jest jakość algorytmów wektoryzacji (szybkość działania, dokładność odwzorowania), a także możliwości późniejszego użycia wczytanego obrazu. Jak wspomniano wyżej, dobrym rozwiązaniem jest traktowanie tych obrazów identycznie jak krojów pisma i dopuszczenie wprowadzania poprawek po procesie wektoryzacji (FolioZnak – VinylWriter). Spotyka się też rozwiązanie polegające na wektoryzacji wczytanego obrazu dopiero na etapie projektowania końcowego obrazu (rys. 1C) i jego przechowywaniu jako elementu tego obrazu (Sign Studio Plus).

- Możliwość poprawiania standardowych krojów pisma oraz tworzenia krojów nowych. Rzadko spotykana cecha (jedynie FolioZnak – VinylWriter), chociaż wydaje się bardzo istotna. Ciekawym rozwiązaniem programu wspomagającego projektowanie znaków graficznych jest edytor ZSoft Type Foundry, zrealizowany w środowisku MS-Windows. Mimo wielu wad (jako edytor ogólnego przeznaczenia nie ma specjalizowanych funkcji przydatnych w zastosowaniach CAS, zły dobór kolorów i niewygodny sposób projektowania), niektóre zastosowane w nim rozwiązania wydają się być godne uwagi.

- Efektywność wspomagania projektowania na etapie C (rys. 1). Istotna jest prostota obsługi, liczba dostępnych przekształceń graficznych, łatwość wprowadzania zmian i poprawek. Istnieją tu różne rozwiązania, które ogólnie podzielić można na te, których podstawą jest zastosowanie urządzenia wskaźnikowego (Sign Studio Plus, Vinyl Cad, oraz Corel Draw, który został w tym miejscu wymieniony jedynie ze względu na ciekawy sposób projektowania, nie ma on bowiem cech typowego systemu CAS) oraz te, które operują głównie przez klawiaturę (FolioZnak – VinylWriter, CSR).

- Dostępność jak największej liczby urządzeń wyjściowych oraz dobra jakość produktu końcowego. Zagadnienie to znajduje się poza zakresem zainteresowania niniejszego artykułu i nie będzie szerzej omawiane.

Dodać należy, że większość istniejących systemów nie zapewnia wspomagania wszystkich etapów projektowania produktu końcowego, a jedynie niektórych. Pozostałe kroki są w takich przypadkach wykonywane za pomocą programów nie należących do danego systemu. Wpływa to niekorzystnie na spójność narzędzia wspomagającego, czego wynikiem są trudności w opanowaniu programów przez użytkownika.



Rys. 2. Przykład złego (a) i dobrego (b) opisu gładkiej krzywizny

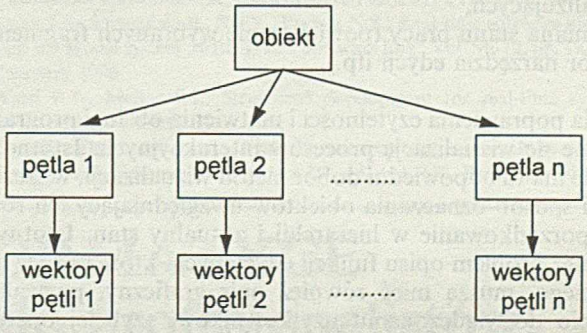
Koncepcja nowego systemu CAS

Istotnymi parametrami określającymi jakość systemów CAS, a ściślej rzecz biorąc ich modułów wspomagających projektowanie, są: wygoda obsługi, sposób obrazowania poszczególnych etapów edycji, sposób zadawania przekształceń graficznych oraz liczba i rodzaj dostępnych przekształceń. Trzy pierwsze cechy mają decydujący wpływ na szybkość pracy podczas korzystania z systemu.

Podstawową sprawą jest wybór sposobu opisu wektorowego obiektów graficznych. Istniejący tu standard ciągu wektorów linii prostych i 4-punktowych krzywych Bézier wydaje się być rozwiązaniem optymalnym. Pozwala na precyzyjne zdefiniowanie kształtu obiektów przy minimalnej liczbie wektorów opisujących. Jednocześnie nie wprowadza zbędnego skomplikowania zagadnienia opisu, co nie jest bez znaczenia dla użytkownika systemu. Każdy znak jest opisany za pomocą linii prostych i segmentów kubicznych Béziera. Krzywe nie są przeciągane poza ekstrema X, Y oraz punkty przegięcia. Istotne z punktu widzenia jakości produktu końcowego jest zapewnienie gładkości krzywizn konturów. Uzyskuje się to przez odpowiednie dobranie punktów wiodących krzywych Bézier opisujących krzywiznę. Gładkość (rozumianą jako istnienie pierwszej pochodnej w punkcie wspólnym dwóch krzywych) uzyskuje się przez takie ustawienie punktów wiodących, aby punkty: przedostatni pierwszej krzywej, ostatni pierwszej krzywej/pierwszy drugiej krzywej i drugi drugiej krzywej, znajdowały się na tej samej prostej. Analogicznie uzyskuje się gładkość połączenia prostej i krzywej Bézier: drugi punkt krzywej musi znajdować się na prostej.

Najwięcej problemów sprawia zwykle implementacja kroku *B* (rys. 1), a więc poprawki obrazu zwektoryzowanego i standardowych krojów pisma. Przy projektowaniu modułów realizujących ten krok, należy:
- zapewnić optymalną jakość opisu kształtu,
- zapewnić optymalizację kierunku i kolejność wektorów,
- wprowadzić jednolity i przejrzysty system obsługi.

Optymalizacja jakości opisu wpływa na polepszenie jakości końcowego produktu, a także przyspieszenie procesu jego wykonania (na etapie przesłania do urządzenia wyjściowego). Patrząc na zagadnienie od strony plotera tnącego, każdy obiekt graficzny jest zbiorem zamkniętych pętli, z których każda składa się z wektorów (por. rys. 3). Istotne jest zapewnienie prawidłowej kolejności pętli, a także optymalne ułożenie wektorów względem siebie tak, aby zapewnić minimalizację ruchów części mechanicznych plotera. Wykonaniu tego zadania służyć może zestaw funkcji optymalizujących, które na podstawie zadanej strategii wejściowej automatycznie dobierają wzajemne ułożenie wektorów tworzących obiekt. Inny zestaw funkcji powinien zajmować się poprawianiem jakości opisu (sprawdzanie i korekty ciągłości połączeń między wektorami, wychwytywanie przeciągania krzywych poza ich ekstrema X, Y i punkty przegięcia itd.). Interesujący jest tu nie tyle problem algorytmów realizujących te funkcje, a zagadnienie styku programów z obsługującym je użytkownikiem. Dopiero złożenie rozwiązań tych dwóch problemów daje rozwiązanie całościowe, w którym coraz

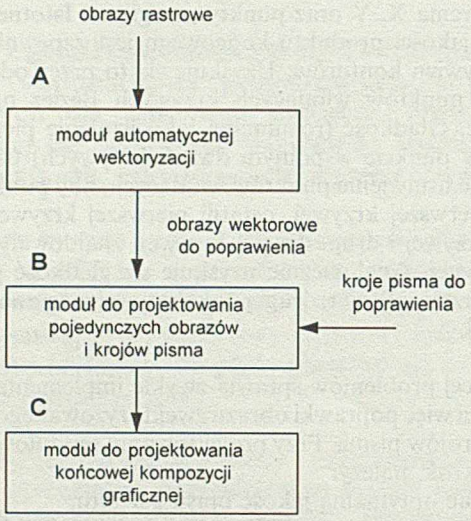


Rys. 3. Struktura opisu obiektu graficznego

większe znaczenie ma zagadnienie optymalizacji procesu obsługi programów.

W celu zoptymalizowania procesu projektowania oraz zunifikowania sposobu podejścia do zagadnienia, należałoby odtworzyć w programie realizującym etap *B* (rys. 1), podaną wyżej strukturę obiektu. Miałoby to tę zaletę, że zbliżałoby sposób zachowania programu do sposobu myślenia użytkownika, który na podstawie swojego dotychczasowego doświadczenia nabiera zwykle określonych nawyków.

Projektowany system ma strukturę blokową zobrazowaną na rys. 4. Struktura ta odzwierciedla przedstawiony na rys. 1 schemat projektowania produktu końcowego, a kolejne moduły odpowiadają kolejnym krokom procesu realizacji.



Rys. 4. Struktura projektowanego systemu

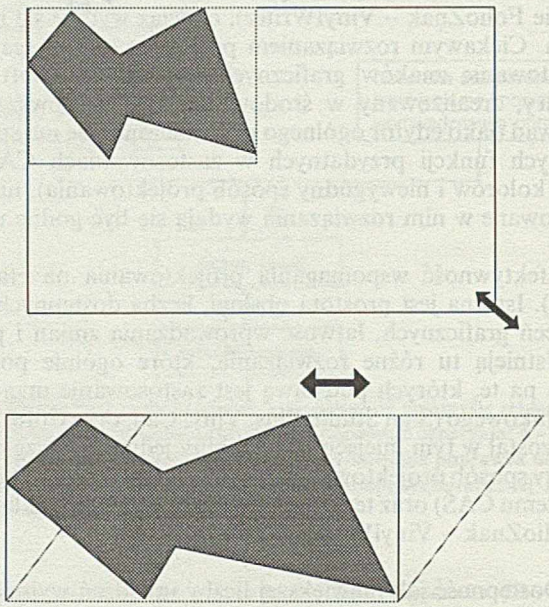
W proponowanym rozwiązaniu zakłada się projektowanie zorientowane obiektowo. Istnieją klasy i hierarchia obiektów (w rozumieniu węższym niż dotychczas, tj. jako prymitywy wejściowe i ich grupy), będące odzwierciedleniem przedstawionej struktury. Obiekty tak zdefiniowane można grupować, poddawać przekształceniom graficznym oraz w dowolny sposób modyfikować. Podejście takie zdaje się być wygodne i logiczne, a także zapewniające optymalny sposób zarówno wprowadzania nowych obiektów, jak i modyfikacji obiektów już istniejących.

Z punktu widzenia kontaktu program – użytkownik, funkcje edycyjne programów wspomagających projektowanie można pogrupować w następujący sposób:

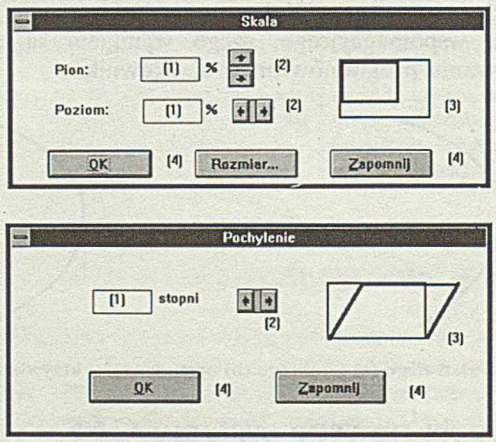
- wprowadzania prymitywów graficznych (prosta, krzywa, wielobok, pętla, tekst itp.),
- manipulacja obiektami (transformacje, sposób ich zadawania itp.),
- wprowadzanie parametrów dla automatycznych funkcji optymalizujących,
- zmiana stanu pracy (powiększanie wybranych fragmentów, wybór narzędzia edycji itp.).

Dla poprawienia czytelności i ułatwienia obsługi programów stosuje się wizualizację procesów interakcyjnych. Istotne znaczenie ma tu odpowiedni dobór metod wizualizacji, w szczególności sposób oznaczania obiektów uwzględniający ich rodzaj, przyporządkowanie w hierarchii i aktualny stan. Istotny jest również problem opisu funkcji edycyjnych, które oprócz opisu słownego, muszą mieć również opis graficzny, pozwalający średnio doświadczonemu użytkownikowi szybciej operować dostępnymi narzędziami edycji oraz umożliwiać przejrzystą wizualizację stanu, w jakim znajduje się program. Podobne znaczenie ma też dobór kształtu, koloru i wielkości kursora, który oprócz funkcji wskaźnika urządzenia wejściowego, przejmuje też funkcję informującą. Innym istotnym problemem jest podział ekranu na obszary funkcjonalne tak, aby był on łatwo czytelny i jednocześnie umożliwiał przekazanie jak największej liczby informacji.

Przedstawione wyżej zagadnienia mają charakter statyczny. Przy projektowaniu występują też zagadnienia wizualizacji procesów dynamicznych, jak np. zadawanie parametrów transformacji, czy bieżące nanoszenie zmian w poszczególnych obiektach. Problemy te mają zasadnicze znaczenie dla jakości narzędzia wspomagającego projektowanie. Dobrym rozwiązaniem wydaje się być opracowanie mechanizmów wizualizacji przekształceń, które z jednej strony układałyby się w spójny system, a z drugiej w charakterystyczny sposób odzwierciedlałyby każdy rodzaj transformacji. Poniższy przykład (rys. 5) ilustruje różne sposoby skalowania i pochylania obiektów.



Rys. 5. Graficzny sposób zadawania skalowania (a) i pochylania (b) obiektów graficznych
a) pomocnicze linie pokazują zarówno nową wielkość obiektu, jak też zmianę w stosunku do starej wielkości
b) pomocnicze linie pokazują aktualne pochylenie obiektu na tle jego pierwotnego wyglądu



Rys. 6. Zadawanie parametrów liczbowych z wizualizacją ich znaczenia: a) dla skalowania, b) dla pochylania

W obu przypadkach zadawanie parametrów przekształcenia odbywa się w oknie dialogowym. Można odpowiednie liczby podać przez ich wpisanie z klawiatury (w polu 1), lub „naciskami" strzałek „więcej, mniej" w polu (2). Jednocześnie w polu (3) pojawia się informacja graficzna obrazująca wynik przekształcenia przy aktualnych parametrach. Potwierdzenie wybranych parametrów lub anulowanie przekształcenia odbywa się przez „naciśnięcie" odpowiedniego pola (4). W tym przypadku skalowanie można rozszerzyć np. o opcję podania docelowych wymiarów (zamiast zadawania współczynników skali).

* * *

W artykule[1)] omówiono przeznaczenie graficznych systemów wspomagania projektowania w zastosowaniach typu CAS i podano zarys koncepcji systemu tego typu ze szczególnym uwzględnieniem optymalizacji komunikacji użytkownik – program. Przedstawiono też problemy wynikające z konstrukcji systemu CAS wraz z przykładowymi rozwiązaniami. Realizacja wybranych fragmentów omówionego systemu odbyła się w ramach pracy dyplomowej w semestrze letnim 1991–1992.

Zagadnienie optymalizacji komunikacji programu z użytkownikiem w pełniejszy sposób przedstawiono w pracy [3], gdzie można też znaleźć przykłady istniejących rozwiązań w różnego typu systemach (nie tylko graficznych). W pracy [1] podano podstawowe informacje o sposobach i wybranych standardach opisu krojów pisma.

1) Przedruk z czasopisma „Machine GRAPHICS & VISION. Vol. 1, No. 1–2, 1992. Wydawca: Instytut Podstaw Informatyki PAN

LITERATURA

[1] Adobe Type 1 font format. Adobe Systems Incorporated, 1990
[2] Draus A., Mazur T.: Corel Draw! Podręcznik Użytkownika, PLJ, 1991
[3] Preece J., Keller L.: Human-Computer Interaction. Prentice Hall, 1989
[4] Sign Studio Plus User Manual. The Electric Studio, 1990
[5] VinylWriter User Manual. Wight Scientific, 1991

Projektowanie oprogramowania systemów wbudowanych

dokończenie ze s. 15

Zasada projektowania obiektowego zyskała powszechne uznanie jako jedno z istotnych kryteriów dekompozycji i strukturalizacji programów. Podejście to jest szczególnie użyteczne przy projektowaniu oprogramowania systemów wbudowanych, zanurzonych w złożonym otoczeniu sprzętowym. W tym kontekście zasady projektowania obiektowego zostały powszechnie zaakceptowane przez projektantów i programistów języka Ada. O efektywności tego podejścia świadczy również rosnąca popularność języków programowania obiektowego, w tym zwłaszcza C++. Z drugiej strony, podejście to nie jest jeszcze poparte dobrze zdefiniowanymi metodami projektowymi, gwarantującymi powtarzalność rozwiązań.

Metody formalne oraz koncepcja transformacyjnej implementacji nie osiągnęły jeszcze poziomu umożliwiającego przemysłowe zastosowanie na szeroką skalę. Ponieważ transformacje stosowane do przekształcenia specyfikacji w implementację muszą mieć dowody poprawności, obydwa podejścia są wyraźnie ze sobą związane. Jak na razie, badania w tych kierunkach biegły oddzielnie, co zapewne jest jednym z powodów niezadowalającego stanu tych metod.

Podsumowując ten krótki przegląd można stwierdzić, że najbardziej pragmatyczne podejście do projektowania oprogramowania opiera się na modelu tradycyjnym, z ewentualnym wykorzystaniem technik szybkiego makietowania. Dodatkową zaletą tego podejścia jest istnienie wielu systemów wspomagających projektowanie (CASE – *Computer Aided Software Engineering*), związanych z najpopularniejszymi metodami specyfikacji i projektowania.

LITERATURA

[1] Alford M.W.: SREM at the age of eight; the distributed computing design system. Computer, Vol. 18, pp. 36–46, April 1985
[2] Booch G.: Object-oriented development. IEEE Trans. Software Eng., Vol. SE-12, pp. 211–221, February 1986
[3] Booch G.: Software engineering with Ada. Benjamin/Cummings, 1987
[4] Cameron J.R.: An overview of JSD. IEEE Trans. Software Eng., Vol. SE-12, pp. 222–240, February 1986
[5] Dijkstra E.W.: A discipline of programming. Prentice-Hall, 1976
[6] DoD-Std-2167A. Defense systemsoftware development, February 1988
[7] Halang W., Sacha K.: Real-Time System. World Scientific, Singapore, New Jersey, 1992
[8] Hoare C.A.R.: The axiomatic basis of computer programming. Commun. ACM, Vol. 12, pp. 576–583, October 1969
[9] Jackson M.A.: Principles of program design. Academic Press, New York, 1975
[10] Jones C.B.: Software development: A rigorous approach. Prentice-Hall International, 1980
[11] Mullery G.P.: CORE – A method for controlled requirements specification. Proc. 4th Int Conf. on Software Eng., pp. 126–135, 1979
[12] Peterson J.L.: Petri net theory and modeling of systems. Prentice-Hall, Inc., 1981
[13] Ross D.T.: Applications and extensions of SADT, Computer, Vol. 18, pp. 25–34, April 1985
[14] Sacha K.: Transformational implementation of PAISLey specifications using Petri nets. Software Eng. Journal, Vol. 7, No 3, 1992, pp. 191–204
[15] Sacha K.: Transnet approach to requirements specification and prototyping. Proc. 6th Annual European Conference „Comp-Euro '92", The Hague, 1992, pp. 220–225
[16] Sievert G.E., Mizell T.A.: Specification-based software engineering with TAGS. Computer, Vol. 18, pp. 56–65. April 1982
[17] Simpson H.: The Mascot method. Software Eng. Journal, Vol. 1, pp. 103–120, May, 1986
[18] Stąpor K.: Strukturalna analiza i projektowanie systemów informatycznych – metodyka Gane-Saesona. INFORMATYKA, nr. 2 i 3, 1992
[19] Tseng J.S., Szymański B., Shi Y., Prywes N.S.: Real-time software life cycle with the Model system. IEEE Trans. Software Eng., Vol. SE-12, pp. 192–197, February 1986
[20] Ward P.T., Mellor S.J.: Structured development for real-time systems. Yourdon Press, 1986
[21] Zave P.: The operational versus the conventional approach to software development. Commun. ACM, Vol. 27, pp. 104–118, February 1984.

Ogłoszenie w INFORMATYCE dotrze przede wszystkim do profesjonalistów!

JAN ZABRODZKI
Instytut Informatyki
Politechnika Warszawska

Grafika komputerowa w programie studiów

Grafika komputerowa jest jednym z najszybciej rozwijających się działów informatyki. Wiąże się to z coraz większym docenianiem roli komunikacji człowiek-maszyna przez kanał wzrokowy człowieka.- Sprzyja temu niewątpliwie postęp w zakresie konstrukcji sprzętu komputerowego i jego dostępność. Zakres zastosowań grafiki komputerowej obejmuje coraz to nowe dziedziny nauki, techniki, sztuki i rozrywki.

W tej sytuacji znajomość zagadnień grafiki komputerowej staje się niezbędnym elementem wykształcenia informatyka i nauczanie w tym zakresie powinno stać się jednym z podstawowych zagadnień w programie kształcenia. Skuteczność i efektywność nauczania jest uzależniona od koncepcji prowadzenia przedmiotu, dostępu do podręczników oraz dostępu do laboratorium wyposażonego w nowoczesny sprzęt komputerowy w konfiguracji przystosowanej do potrzeb grafiki komputerowej. W przypadku kształcenia inżynierów informatyków istotne jest także zapewnienie dostępu do laboratorium sprzętowego, pozwalającego poznać od strony praktycznej zagadnienia związane z konstrukcją sprzętu wykorzystywanego w zastosowaniach związanych z grafiką komputerową.

Celem artykułu jest przedstawienie koncepcji nauczania grafiki komputerowej oraz problemów związanych z realizacją kształcenia w tym zakresie.

Zakres i przeznaczenie przedmiotu

Grafika komputerowa jest jednym z trzech działów informatyki związanych z obrazem. Inne działy to: przetwarzanie obrazów i rozpoznawanie obrazów. Granice podziału między tymi działami można określić następująco: grafika komputerowa zajmuje się generowaniem (syntezą) obrazów, przetwarzanie obrazów wiąże się z przekształcaniem już istniejących obrazów, natomiast rozpoznawanie obrazów zajmuje się analizą obrazów. O ile elementem łączącym te działy jest obraz, to każdy z tych działów wykorzystuje inne metody i narzędzia. Ze względu na specyfikę poszczególnych działów, próby ich nauczania w jednym przedmiocie na ogół nie prowadzą do dobrych rezultatów. Z drugiej strony wskazane jest, żeby podstawowemu wykładowi z grafiki komputerowej towarzyszyły odrębne wykłady poświęcone pozostałym dwom działom.

Grafika komputerowa w najszerszym ujęciu obejmuje trzy zagadnienia: algorytmy, oprogramowanie i sprzęt. Pełny kurs poświęcony grafice komputerowej powinien podejmować problematykę związaną z każdym z tych trzech zagadnień.

[1] Przedruk z czasopisma „Machine GRAPHICS & VISION. Vol. 1, No. 1–2, 1992. Wydawca: Instytut Podstaw Informatyki PAN

Przedmiot jest adresowany do studentów kierunku informatyka, a więc do przyszłych projektantów systemów grafiki komputerowej, a także twórców specjalistycznego oprogramowania dla określonych zastosowań. Należy tu wyraźnie podkreślić różnicę w stosunku do kształcenia na innych kierunkach studiów, gdzie nauczanie informatyki, a w szczególności grafiki komputerowej sprowadza się do nabycia umiejętności w korzystaniu z gotowych systemów i narzędzi. W szczególności dla specjalistów z innych dziedzin istotne mogą być systemy CAD oraz programy pozwalające na wizualizację wyników pomiarów lub obliczeń.

Koncepcja nauczania

Podstawowe cele nauczania w zakresie grafiki komputerowej można sformułować następująco:

- poznanie podstawowych pojęć, metod i algorytmów,
- nabycie umiejętności praktycznej realizacji systemów i programów grafiki komputerowej,
- poznanie problemów sprzętowych specyficznych dla zaawansowanej grafiki komputerowej,
- poznanie wybranych systemów graficznych (programowych i sprzętowych).

Cele te można osiągnąć prowadząc odpowiednie wykłady, laboratoria i projekty. Studentom szczególnie zainteresowanym grafiką komputerową można oferować odpowiednio zorganizowane pracownie problemowe oraz stwarzać możliwość realizacji prac dyplomowych.

Ze względu na różny stopień zainteresowania studentów problematyką grafiki komputerowej należy przyjąć metodę sekwencyjnego kształcenia, zakładając, że po każdym etapie (w szczególności po pierwszym obowiązkowym) student może zrezygnować z kontynuowania studiowania tej dziedziny. Pełna sekwencja może zawierać następujące etapy:

I. Wykład podstawowy z laboratorium (obowiązkowy, w wymiarze 2 – 2 – każdy).
II. Wykład zaawansowany wraz z laboratorium (obieralny, w wymiarze 2 – 2 – każdy).
III. Wykłady uzupełniające z laboratoriami (obieralne, w wymiarze 2 – 2 – każdy).
IV. Pracownia problemowa.
V. Praca dyplomowa.

Zajęcia związane z etapami II, III i IV, mogą odbywać się sekwencyjnie bądź równolegle.

Wykład podstawowy ma zadanie wprowadzenia (możliwie szerokiego) w problematykę grafiki komputerowej. Towarzyszące wykładowi laboratorium ma umożliwić praktyczne poznanie podstawowych algorytmów grafiki komputerowej.

Wykład zaawansowany (etap II) powiniwn stanowić uzupełnienie wykładu podstawowego. Powinien być poświęcony głów-

nie grafice 3D oraz prezentować wybrane zastosowania grafiki. Wykład ten nie musi mieć ściśle ustalonego programu. W ramach laboratorium powinien być realizowany duży projekt (semestralny) związany z treścią wykładu.

Wszystkie wykłady uzupełniające (etap III) powinny być realizowane w wymiarze 2 – 2 – każdy. Każdy z tych wykładów musi być tak prowadzony, aby umożliwiał korzystanie z innych wykładów uzupełniających. Wykłady te powinny umożliwiać szczegółowe poznanie wybranych zagadnień grafiki komputerowej i działów pokrewnych. W szczególności wśród tych przedmiotów mogą się znaleźć:

- modelowanie geometryczne,
- systemy CAD/CAM,
- wizualizacja danych,
- graficzne bazy danych,
- geometria obliczeniowa,
- systemy edycyjne,
- przetwarzanie obrazów,
- rozpoznawanie obrazów,
- architektury systemów graficznych,
- układy VLSI dla potrzeb grafiki komputerowej.

W ramach pracowni problemowych studenci – współpracując z opiekunami naukowymi – mają możność indywidualnej realizacji projektów z zakresu grafiki komputerowej. Realizację pracowni problemowej powinno ułatwić uczęszczanie na odpowiednie przedmioty uzupełniające. Praca dyplomowa wiąże się z rozwiązaniem określonego tematu badawczo-konstrukcyjnego (programowego lub sprzętowego). Niezbędnym warunkiem realizacji wszelkich zajęć grafiki komputerowej jest możliwość korzystania z odpowiednio wyposażonego laboratorium.

Przedstawiona koncepcja układu przedmiotów z zakresu grafiki komputerowej umożliwia łatwe dostosowanie programu do zainteresowań studentów, jak też do potrzeb i możliwości realizacyjnych różnych uczelni. Przy pełnej realizacji tej koncepcji kształcenia student ma możność wyboru zakresu i stopnia szczegółowości studiowania poszczególnych zagadnień. Etap III pozwala na stosunkowo łatwe i elastyczne dostosowywanie programów nauczania do zmieniających się zainteresowań studentów oraz szybkiego rozwoju grafiki komputerowej.

Uczelnia dysponująca skromniejszymi możliwościami może ograniczyć się do prowadzenia jedynie niektórych etapów nauczania – w krańcowym przypadku jedynie przedmiotu podstawowego. Natomiast studenci szczególnie zainteresowani tą problematyką mogą korzystać z bogatszej oferty proponowanej w innej uczelni, realizującej program w szerszym zakresie (zwłaszcza z zasobów lepiej wyposażonego laboratorium grafiki komputerowej).

Podstawowy wykład

Wykład z grafiki komputerowej powinien znajdować się w grupie przedmiotów podstawowych dla studentów kierunku informatyka. Powinien on być umieszczony w jednym z semestrów od 4 do 7 (przy studiach trwających 10 semestrów). Wynika to stąd, że z jednej strony konieczne jest wstępne przygotowanie studenta, a z drugiej strony potrzebny jest czas na prowadzenie przedmiotów uzupełniających i rozszerzających, które powinny być wykładane później. Istotne jest również i to, aby wiadomości z zakresu grafiki komputerowej mogły być wykorzystywane w innych przedmiotach, zwłaszcza projektowych (pracownie problemowe, prace dyplomowe). Jeżeli w konkretnym programie studiów nie przewiduje się kontynuacji przedmiotu, to może być on umieszczony również na późniejszych semestrach. Jeżeli program studiów zakłada podział przedmiotów na obowiązkowe i obieralne, to należy przyjąć, że podstawowy wykład powinien znajdować się w grupie przedmiotów obieralnych.

Jak wspomniano wyżej, przedmiot „Grafika komputerowa" wymaga wcześniejszego opanowania określonych wiadomości. Chodzi tu głównie o odpowiednie przygotowanie z zakresu matematyki, programowania, teorii algorytmów, architektur maszyn cyfrowych.

W wykładzie podstawowym o ograniczonym czasie trwania (30 godzin) możliwe jest omówienie jedynie wybranych zagadnień. Przy opracowywaniu konspektu przedmiotu trzeba zachować kompromis między szczegółowością omawiania poszczególnych zagadnień, a ogólnym zakresem poruszanych problemów. Proponowany zestaw tematów omawianych w trakcie wykładu obejmuje następujące zagadnienia:

- wprowadzenie do techniki rastrowej,
- podstawowe wiadomości o sprzęcie grafiki komputerowej,
- podstawowe algorytmy (kreślenie odcinka i okręgu, wypełnianie, obcinanie, transformacje itp.),
- kolor,
- eliminacja zakłóceń,
- modelowanie obiektów 3D,
- krzywe i powierzchnie Bezier, B-spliny,
- powierzchnie niewidoczne,
- modele oświetlenia powierzchni,
- metoda śledzenia promieni, metoda energetyczna,
- tekstura,
- fraktale,
- animacja,
- standardy,
- architektury systemów graficznych.

Towarzyszące wykładowi laboratorium powinno umożliwić praktyczne poznanie realizacji podstawowych algorytmów i technik stosowanych w grafice komputerowej. Istotne jest zwrócenie uwagi na ograniczenia i niedoskonałości techniki rastrowej, przenośność oprogramowania oraz sposób organizacji współpracy z użytkownikiem. Zajęcia laboratoryjne powinny być realizowane w formie kilku niedużych problemów projektowych.

Laboratorium grafiki komputerowej

Niezbędnym elementem w procesie kształcenia w zakresie grafiki komputerowej jest właściwie wyposażone laboratorium. Wymagane wyposażenie laboratorium zależy od etapu, na którym są prowadzone określone zajęcia.

Dla prowadzenia zajęć na poziomie podstawowym wystarczające jest wyposażenie laboratorium w sprzęt klasy IBM AT z kartami graficznymi o rozdzielczości 640 × 480 i minimum 16-elementowej palecie kolorów (np. VGA, SVGA) oraz w odpowiednie monitory kolorowe. Liczba komputerów powinna być na tyle duża, aby każdy student mógł samodzielnie uruchamiać programy realizujące podstawowe algorytmy grafiki 2D i 3D.

Wykorzystywany do zajęć podstawowych sprzęt może być w zasadzie wykorzystywany do zajęć na etapie II, przy realizacji programów dotyczących bardziej złożonych zagadnień grafiki komputerowej. Przy realizacji niektórych algorytmów może się okazać konieczne udostępnienie kart i monitorów o większej rozdzielczości (1024 × 768) i liczbie kolorów (minimum 256).

Zajęcia zaliczone do etapu III wymagają wyposażenia laboratorium w bardziej różnorodny i specjalizowany sprzęt. Nie musi on być dostępny w dużych ilościach ze względu na mniejszą liczbę studentów biorących udział w tych zajęciach. W szczegól-

ności należy wyposażyć laboratorium w różnego rodzaju urządzenia zewnętrzne do wprowadzania i wyprowadzania informacji graficznej (skaner, tablet, pisak XY, drukarka kolorowa, kamera kolorowa, frame-grabber, magnetowid, naświetlarka kliszy fotograficznej itp.).

Do zajęć w ramach pracowni problemowych i prac dyplomowych niezbędne jest wyposażenie laboratorium w niezbyt liczne urządzenia pozwalające realizować projekty na poziomie profesjonalnym (bądź zbliżonym do profesjonalnego). W szczególności w laboratorium powinny się znajdować systemy z kartami kolorowymi o wysokiej rozdzielczości i palecie kolorów 2^{15} lub 2^{24}. Laboratorium powinno być wyposażone co najmniej w jedną profesjonalną stację graficzną.

Jest oczywiste, że do prac prowadzonych w laboratorium powinno być wykorzystywane wyłącznie licencjonowane oprogramowanie. Laboratorium powinno mieć zestaw profesjonalnych pakietów związanych z różnymi zastosowaniami grafiki komputerowej.

W przypadku uczelni technicznej, laboratorium powinno być również wyposażone w odpowiedni zestaw montażowo-pomiarowy, który umożliwi praktyczną weryfikację różnych koncepcji sprzętowo-architektonicznych oraz realizację wybranych fragmentów systemów z wykorzystaniem współczesnych układów VLSI, dedykowanych dla zastosowań w systemach graficznych.

Przedstawione omówienie problemu nauczania w zakresie grafiki komputerowej pokazuje, jak wiele problemów wiąże się z dobrym nauczaniem tej specjalizacji. W szczególności bogactwo tematyczne grafiki komputerowej i związanych z nią innych działów informatyki wskazuje na konieczność wyodrębnienia kształcenia podstawowego oraz pogłębiania wiedzy w dalszych przedmiotach obieralnych. Niezbędnym warunkiem dla efektywnego kształcenia jest dostęp do dobrze wyposażonego laboratorium. Istotne jest również dobre wyposażenie w środki audiowizualne (rzutniki pisma, przezroczy, RGB). Ważne jest również zapewnienie studentom dostępu do odpowiedniej literatury – podręczników (w języku polskim i angielskim) oraz czasopism naukowych.

LITERATURA

[1] Burton R.P.: Advanced university courses in computer graphics. Computer Aided Design 1989, No. 2, pp. 116–119

[2] Cunningham R.S.: Computer graphics in computing curriculum 91. Computer Graphics 1991, No. 3, pp. 208–210

[3] Nauczania grafiki komputerowej na kierunku informatyka. Raport wewnętrzny IIPW, 1991

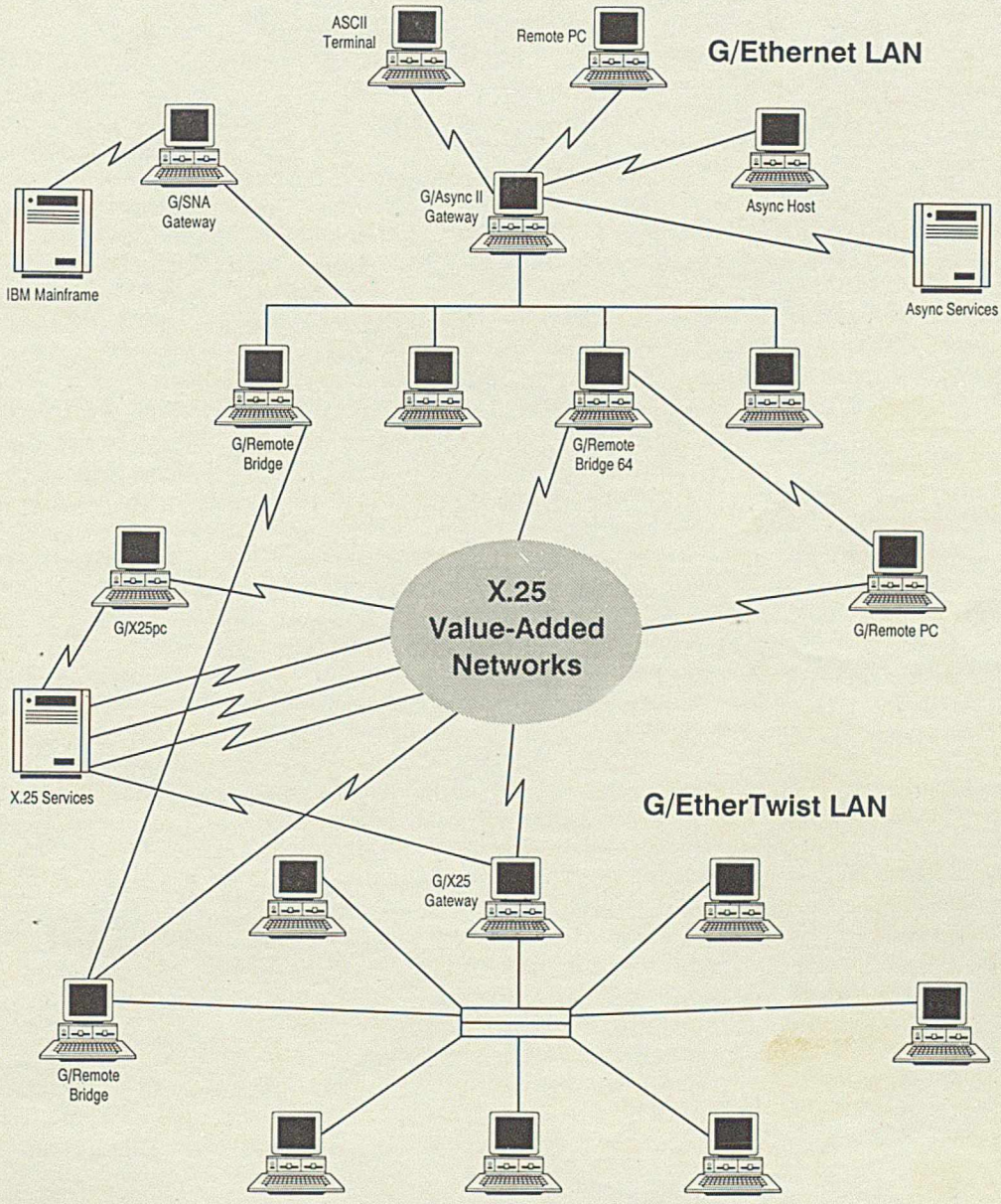
[4] Owen G.S.: Visualization education in the USA. IFIP WG3.2 – Working Conference. Zurich 1991.

Profesor JAN ZABRODZKI kierownik Zakładu Grafiki Komputerowej w Instytucie Informatyki Politechniki Warszawskiej. Zainteresowania naukowe są związane z szeroko rozumianą grafiką komputerową, zwłaszcza w zakresie systemów wizualizacji w czasie rzeczywistym oraz metodami projektowania i testowania urządzeń i systemów cyfrowych.

Zieliński K., Pardyak P.: Emerald – język i system rozproszonego programowania obiektowego INFORMATYKA 1993, nr 7, s. 1 Charakterystyka języka Emerald, łączącego dwie nowoczesne dziedziny informatyki: programowanie obiektowe i systemy rozproszone.	**Zieliński K., Pardyak P.: Emerald – a language and system for object distributed programming** INFORMATYKA 1993, No. 7, p. 1 Characteristics of the Emerald language, which joins two modern informatic disciplines: object oriented programming and distributed systems.	**Zieliński K., Pardyak P.: Emerald – eine Sprache und System der verteilten Objektprogrammierung** INFORMATYKA 1993, Nr. 7, S. 1 Eine Charakteristik von der Emerald-Sprache, die zwei moderne Informatikbereiche: objektorientierte Programmierung und verteilte Systeme, vereinigt.
Sacha K.M.: Projektowanie oprogramowania systemów wbudowanych INFORMATYKA 1993, nr 7, s. 8 Przegląd i porównanie różnych współczesnych metod projektowania oprogramowania.	**Sacha K.M.: Designing of software for embeded systems** INFORMATYKA 1993, No. 7, p. 8 Survey and comparison of different contemporary methods for software designing.	**Sacha K.M.: Projektierung der Software für eingebaute Systeme** INFORMATYKA 1993, Nr. 7, S. 8 Ein Übersicht und Vergleich von verschiedenen zeitgenössischen Methoden der Softwareprojektierung.
Goliński M.: Rynki elektroniczne – podstawowe pojęcia, obszary zastosowań i aspekty ekonomiczne INFORMATYKA 1993, nr 7, s. 16 Charakterystyka rozwoju rynków elektronicznych oraz przykłady już zrealizowanych elektronicznych struktur rynkowych.	**Goliński M.: Electronic markets – basic conceptions, application areas and economic aspects** INFORMATYKA 1993, No. 7, p. 16 Characteristics of electronic markets development and examples of realized electronic market structures.	**Goliński M.: Elektronische Märkte – Grundbegriffe, Anwendungsbereiche und ökonomische Aspekte** INFORMATYKA 1993, Nr. 7, S. 16 Eine Charakteristik von Entwicklung der elektronischen Märkte und Beispiele der schon realisierten elektronischen Marktstrukturen.
Grzelak T.: Systemy wspomagania projektowania w przemyśle reklamowym INFORMATYKA 1993, nr 7, s. 22 Charakterystyka istniejących rozwiązań systemów typu CAS (Computer Aided Signmaking) oraz koncepcja nowego systemu tego typu.	**Grzelak T.: Assisted system design in advertising industry** INFORMATYKA 1993, No. 7, p. 22 Characteristics of existed CAS (Computer Aided Signmaking) system solutions and the idea of a new similar system.	**Grzelak T.: Systeme der unterstützten Projektierung in der Reklameindustrie** INFORMATYKA 1993, Nr. 7, S. 22 Eine Charakteristik von heutigen Lösungen der CAS (Computer Aided Signmaking)-Systemen und eine Konzeption eines neuen Systems dieser Art.
Zabrodzki J.: Grafika komputerowa w programie studiów INFORMATYKA 1993, nr 7, s. 26 Koncepcja nauczania grafiki komputerowej w wyższej uczelni technicznej.	**Zabrodzki J.: Computer graphics in university studies program** INFORMATYKA 1993, No. 7, p. 26 The idea of computer graphics education in technical university.	**Zabrodzki J.: Computergrafik im Studienprogramm** INFORMATYKA 1993, Nr. 7, S. 26 Eine Konzeption der Computergrafikausbildung in einer technischen Universität.

Produkty sieciowe LAN i WAN

Gateway communications, inc.



BEZKONKURENCYJNE PRODUKTY
SIECI KOMPUTEROWYCH NAGRADZANE PRZEZ:

PC MAGAZINE, LAN MAGAZINE, INFO WORLD

OFERUJE AUTORYZOWANY DYSTRYBUTOR:

MIKROB

PC MAGAZINE EDITORS' CHOICE January 31, 1989 G/Ethernet AT

PC MAGAZINE EDITORS' CHOICE October 16, 1990 G/EtherTwist

P.W.P.T. MIKROB SP. Z O.O.

20-346 Lublin, ul. Długa 5
Tel. (0-81) 420-61, faks 415-43, teleks 643776 MIKRO PL

0/15/91

Na życzenie wysyłamy katalog produktów. Atrakcyjny program współpracy dla dealer'ów.

JUNISOFTEX®

Przedsiębiorstwo Produkcyjno-Usługowe Innowacji Technicznych Biuro Techniczno-Handlowe 44-100 Gliwice ul. Konstytucj 11.
tel. 31-57-00, 31-51-31, 31-75-10, fax 31-75-10, centr. 31-90-81 do 88, telex 036233
Zespół wdrożeń i analiz systemów komputerowych wewn. 15 ,25, 282, Zespół inżynierii oprogramowania wewn. 14, 20, 245,
Zespół d/s serwisu systemów Novell i sprzętu komputerowego wewn. 16, 245, Dział Handlowy wewn. 18, 28, 201,
Szkoła Komputerowa wewn. 282, Sekretariat wewn. 250, Prezes wewn. 13, 272, Dyrektor wewn. 12, 272, Główny Księgowy wewn. 256,
** Konto: PKO I/O Gliwice 27661-225298-136 **

OFERUJEMY

Kompleksową komputeryzację przedsiębiorstw obejmująca dostawę i wdrożenie zintegrowanych systemów komputerowych takich jak system:

finansowo-kosztowy (automatyczne rozliczanie kosztów)
+
gospodarka materiałowa
+
środki trwałe
+
wyposażenie
+
obrót towarowy fakturowanie sprzedaży
+
kadry-płace
+
techniczne przygotowanie produkcji

Sprzęt komputerowy w tym:

Komputery
Monitory
Dyski twarde
Drukarki (mozaikowe i laserowe)
osprzęt sieciowy (karty sieciowe, kable zwykłe i światłowodowe)

Systemy i oprogramowanie licencjonowane

DOS
Novell
Btrieve
SQL Base
Windows

Szkolenia komputerowe

podstawy obsługi komputerów
arkusze kalkulacyjne (Qpro, Lotus, EXCEL)
edytory tekstów (WordPerfect, TAG, ChiWriter)
bazy danych (dbase IV, Clipper, Paradox)
systemy operacyjne (DOS, Novell)

SOFTWARE SUPPORT FOR DOS, NOVELL 2.2, 3.11, 4.0, SQL BASE firmy GUPTA

*

OPROGRAMOWANIE DLA PRZEDSIĘBIORSTW

oferujemy własnej produkcji zakładowy system informatyczny PHU-Perfect(N) obejmujący swoim zakresem:
system finansowo-kosztowy umożliwiający automatyczne rozliczanie kosztów, atomatyczną dekretację, analizę rozrachunków
system obrotu materiałami, wyrobami, towarami (kody paskowe) w tym: stany magazynowe, ewidencje zamówień, fakturowanie sprzedaży (VAT) z automatyczną dekretacją rozdzielników kosztów i sprzedaży do systemu finansowo-kosztowego,
system ewidencji majątku trwałego i wyposażenia
system kadrowo-płacowy (SQL) z automatyczną dekretacją rozdzielników płacowych do systemu finansowo-kosztowego
system technicznego przygotowania produkcji (uk. 1993 r)
(oprogramowanie w systemie SQL BASE w przygotowaniu)

*

OPROGRAMOWANIE DLA URZĘDÓW MIAST I GMIN

oferujemy własnej produkcji system finansowo-podatkowy UMG-Perfect(N) obejmujący swoim zakresem:
system finansowo-księgowy,
systemy podatkowe obejmujące: ustalanie wymiarów, naliczanie i kontrolę płatności należności podatkowych dla dowolnych podatników i/lub płatników podatków na ich kontach i kontach dochodów ze swobodną korektą dowolnych informacji w dowolnym okresie a w tym:
ewidencja zapłat z wykorzystaniem kodów paskowych, emisja wezwań do zapłaty, naliczanie odsetek.
system ewidencji majątku

*

Zapraszamy do współpracy w zakresie wdrożeń naszych systemów partnerów, - firmy informatyczne z całego kraju !!!

0/7/92